FINAL DEGREE PROJECT

TITLE: Smart clinical alarms service for healthcare professionals

DEGREE: IT Engineering Degree

AUTHOR: Rafael Mulero Vellido

ACADEMIC DIRECTOR: Xavier Masip Bruin

ENTERPRISE DIRECTOR: David Vidal

DATE OF PRESENTATION: July, 2017

LAST NAME: Mulero Vellido **NAME:** Rafael

DEGREE: IT Engineering Degree

CURRICULUM: 2010

DIRECTOR: Xavier Masip Bruin

DEPARTMENT: Computer Architecture

PROJECT QUALIFICATION

EXAMINING BOARD

PRESIDENT	SECRETARY	CHAIRPERSON
Jordi García Almiñana	Eva Marin Tordera	Ariadna M. Llorens García

DATE OF PRESENTATION: July 10th, 2017

This project takes into account environmental issues: [ ]**Yes** [x]**No**

RESUM

En aquest Treball de Final de Grau (TFG) es mostra el procés de creació i implementació d'un servei que tindrà com a objectiu informar als professionals mèdics sobre l'estat dels seus pacients en temps real, accelerant el temps de reacció en els casos en què la salut d'aquests necessiti d'una actuació immediata.

El servei seguirà un model client-servidor on el servidor rebrà dades clíniques sobre els pacients hospitalitzats i, en cas de què aquestes dades indiquin algun estat greu o anormal de salut, generarà una alarma que serà enviada a l'aplicació client instal·lada a l'smartphone dels professionals mèdics assignats als pacients.

Aquest TFG s'ha dut a terme a l'Hospital Clínic i Provincial de Barcelona (HCPB) com a proposta del Departament de Sistemes d'Informació (DSI) del mateix hospital amb l'objectiu de crear un servei útil per l'àmbit sanitari.

Paraules clau (màxim 10):

Hospital	Sistema d'Alarmes Clíniques	Sistema de Suport a la Decisió Clínica	Smartphone
Ehealth	Android	Web	

ABSTRACT

The Final Degree Project (FDP) presented below shows the process of creation and implementation of a service that will aim to inform physicians about the status of their patients in real time, speeding up the reaction time in cases where the health of these need immediate action.

The service will follow a client-server model where the server will receive inpatient clinical data and, in case that this data gives signs of some critical or abnormal health status, will generate an alarm that will be sent to the client application installed on the smartphone from the physicians who are in charge of the patients.

This FDP has been done in Hospital Clínic i Provincial de Barcelona (HCPB) as a proposal from the Information System Department (DSI) of the hospital itself with the goal of creating a useful service for the healthcare sector.

Keywords (10 maximum):

Hospital	Clinical Alarm System	Clinical Decision Support System	Smartphone
Ehealth	Android	Web	

Acknowledgement

I would like to thank all the people that have been involved, directly or indirectly, in this journey that has last for seven years.

Thanks to Hospital Clinic i Provincial de Barcelona for giving me the opportunity of creating this project being part of the institution. Specially I would like to thank to David Vidal and Santiago Iriso, director and physician from the Information System Department from the hospital, for helping me to carry on this project and for all the support given during these months.

Thanks to all the UPC professors for all the knowledge I have acquired during all these years. Specially I would like to thank to Xavi Masip, co-director of this project, for all the opportunities that he has offered me for building my career.

Thanks to my family for being there in all the good and bad moments. Specially to my parents and my girlfriend, whom I love with all of my heart.

Finally, I would also like to thank to all my colleagues from all the promotions in which I have been into UPC and to Carlos, Felipe, Ferran, Jessica and Olga from Hospital Clinic for making this journey a little less tough than it could have been.

Table of contents

List of Figures

List of Tables

1 Introduction

The management of information in medical area has undergone a great change in the last years, trying increasingly to change from physical to digital format (digitisation of clinical history). However, the speed with which this information arrives to the destination it should be improved, especially in cases where patients' life depends on, for example, a laboratory result. Nowadays, the amount of time elapsed between the publication of a laboratory result and the arrival of this result to the physician can vary between minutes, hours or, in the worst cases, days and the time is many times very relevant for patient health. To solve this gap of time, there is what is known as Clinical Alarms Systems (CAS), which allows showing relevant information or alarms to physicians when the health of a patient has changed or get worse. The main problem of CAS is that it may send too many alarms, not all of them being really important, making healthcare professionals ignore the alarms because they are being interrupted constantly.

Despite the use of CAS, once the information arrives to the physician, has to review and check if patient status is right or not. This check up can be easy if the amount of data is not very long, but as the amount increases or the information has to be cross-checked with other results it is easy to miss some information, which may increase the response time of the physician. To avoid this cases, there is what is known as Clinical Decision Support Systems (CDSS), which allows processing patient data to obtain different results that will be used in order to help or support the physician to be more efficient with his medical acts. Apparently CDSS seems to be very useful, but they lack of a system to make this information arrive to the final destination in a short time period.

At the same time, we are living in the age where mobile technology has become a key element in the day-to-day life, either for personal or professional area, and we are able to stay connected to the world from anywhere and to send and receive information in a matter of seconds. This FDP tries to enhance CDSS by using mobile technology as CAS in order to improve the response time of healthcare professionals in case of a potential risk for patients.

1.1 Objectives

The following list shows the general and specific objectives that will be tried to achieve by realizing this FDP.

General objectives

- Design and develop the service mentioned above
- Implement the service in a real-world environment

Specific objectives

- Learn new concepts of IT systems
- Improve knowledge on server and mobile technologies
- Learn new working methodologies
- Learn how to correctly manage a project
- Learn how to use LaTeX for writing documents

1.2 Motivation

The motivation of this FDP comes from the need of reducing physicians' response time when the status of a patient is (or can become) critic, improving the quality of clinical service and patients' security. The project has been proposed by the Information Systems Department (Departament de Sistemes d'Informació) from Hospital Clínic i Provincial de Barcelona.

1.3 Area of application

The main idea of the project was to create a system that could be easily applied to any medical centre, but after the analysis and documentation reading done in order to check if this approach would be feasible, the conclusion was that it was very difficult to achieve this goal because of the high difference between the systems of each medical centre. That said, the service has been designed to be used by Hospital Clínic i Provincial de Barcelona services, concretely the emergency and internal medicine services. However, the system has been designed to be extensible, allowing continuing working on it to adapt it to more services inside the hospital.

1.4 State of the art

This section shows a brief description of the state, the evolution and some studies about the different technologies and products used to have a clear idea when developing the project.

1.4.1 Clinical Decision Support Systems

First CDSS were created in the 70-80's thanks to the evolution of Information Technology applied to business [1]. First systems, known as **Knowledge-Based CDSS**, were very rigid and their basic functionality was to give a concrete diagnose based on data introduced by users. Later, in the 90-2000's, they became more flexible and they were able to offer some recommendations or possible diagnoses that the user had to check. In these CDSS, the intelligence of the system consists on a set of instructions that check if the introduced data generates an alarm.

The development of artificial intelligence and artificial neuronal networks [2] helped to create **Nonknowledge-Based CDSS** which differs from the previous ones in the fact that the intelligence of the system is not based on instructions because the system "learns" which results are expected as the user introduces different data. This type of CDSS is more complex but gives results with a higher precision than Knowledge-Based ones.

Some examples of CDSS:

- **VisualDx [3]:** It's mainly used for diseases that present visual symptoms (affections on the skin, eyes...). It's based on Java and uses a big database combining images and text. A software license can be purchased paying a monthly (39'99$/month) or yearly (399'99$/year) subscription or it can be tried for free during 30 days.
- **DXplain [4]:** Very used in academic and professional scope because its ease of use. The data is introduced using plain medical language and the system returns the results. The software is web-based and it can only be used by institutions that purchase a license.
- **Isabel [5]:** Similar to DXplain, allows getting results from information introduced by users, but, in addition, allows to get a deeper knowledge of the given results. It's also web-based software and there are two types of subscription: Standard (199-299$/year) and Premium (329-399$/year).

1.4.2 Clinical Alarms Systems

About CAS there are different articles and studies about the impact of this solution over the performance of the work on the hospital. In November 2015, a study of Colorado University concluded that 98.9% of the alarms generated by a system controlling the effects of opioid were not clinically relevant [6]. On the other hand, a study from Marshfield Clinic made on April 2015 concluded that applying some kind of logic on the alarms generated by the system, made them effective in 80% of cases [7].

Example of smart CAS:

- **Smart Alarms for Medical Device Systems [8]:** This is a project from Pennsylvania University which aims to reduce the amount of alarms generated by a hospital Intensive Care Unit (ICU). To test the system, they focused on patients who were operated of a coronary bypass. The results were successful as the amount of alarms reduced in more than 50%.

1.4.3 State of the art conclusions

The conclusions extracted from the previous cases have served to have conscience that an alarm system without any kind of logic, intelligence or data processing generates too much "noise" because of the lack of relevance of the obtained alarms, making that professionals will end up obviating them. This FDP will try to combine a CAS with a Knowledge-Based CDSS in order to generate a set of alarms that will be useful for professionals without generating too much noise.

1.5 Document structure

This FDP is structured in seven sections, including this **introduction** section. Section 2 presents the **planning** done to build the project. Section 3 analyses the **requirements** the project needs. Sections 4, 5 and 6, more technical, focus on **design**, **development** and **validation** of the system. Finally, section 7 shows the **conclusions** extracted from this FDP. There are also secondary sections like bibliography, list of figures and tables, acronyms...

2 Project planning

2.1 Work packages

The project has been divided in six Work Packages (WP) based on the kind of work taken in each one. The defined WP, detailed in the next subsections, are the following:

- WP1 - Project management
- WP2 - Technical and user requirements
- WP3 - System design
- WP4 - System development
- WP5 - Validation and testing
- WP6 - Project documentation

2.1.1 WP1 - Project management

Package focused on the revision of the project planning and the control of the rest of the tasks from the project. This package has only one task:

- **T1.1 - Review of project tasks and planning**

2.1.2 WP2 - Technical and user requirements

Package focused on the definition of everything that is necessary to make the project, that is, know what are the user needs, which infrastructure will be used... This package contains two tasks:

- **T2.1 - User requirements:** gather information about what the users expect about interfaces, security, ease of use...
- **T2.2 - Technical requirements:** define which technology and infrastructure will be used for the project development.

2.1.3 WP3 - System design

Package focused on the definition of every part of the system. This package contains three tasks:

- **T3.1 - Architecture:** definition of the system architecture.
- **T3.2 - Server design:** definition of the server side components of the system.
- **T3.3 - Client design:** definition of the client side components of the system.

2.1.4 WP4 - System development

Package focused on the development of the components defined in system design. This package contains two tasks:

- **T4.1 - Server development:** development of server side components.
- **T4.2 - Client development:** development of client side components.

2.1.5 WP5 - Validation and testing

Package focused on the testing and validation of each part of the system. This package contains three tasks:

- **T5.1 - Server testing:** validate the correct functionality of the server side.
- **T5.2 - Client testing:** validate the correct functionality of the client side.
- **T5.3 - Integration testing:** validate the correct functionality of both parts of the system working together.

2.1.6 WP6 - Project documentation

Package focused on the documentation that has to be done during the project. This package contains two tasks:

- **T6.1 - Write of project document**
- **T6.2 - Write of initial report**
- **T6.3 - Write of project document abstract**

2.2 PERT diagram

Project Evaluation and Review Techniques (PERT) diagram is used to graphically represent the relationship between the tasks of a project. Figure 2.1 shows the relationship between the previously described Work Packages. WP 1 and 6 are done in parallel during the project. **WP1** has to be done continuously to control that the project is being done according to the planning and **WP6** is highly recommended doing it while the project advances to make sure that everything is documented.

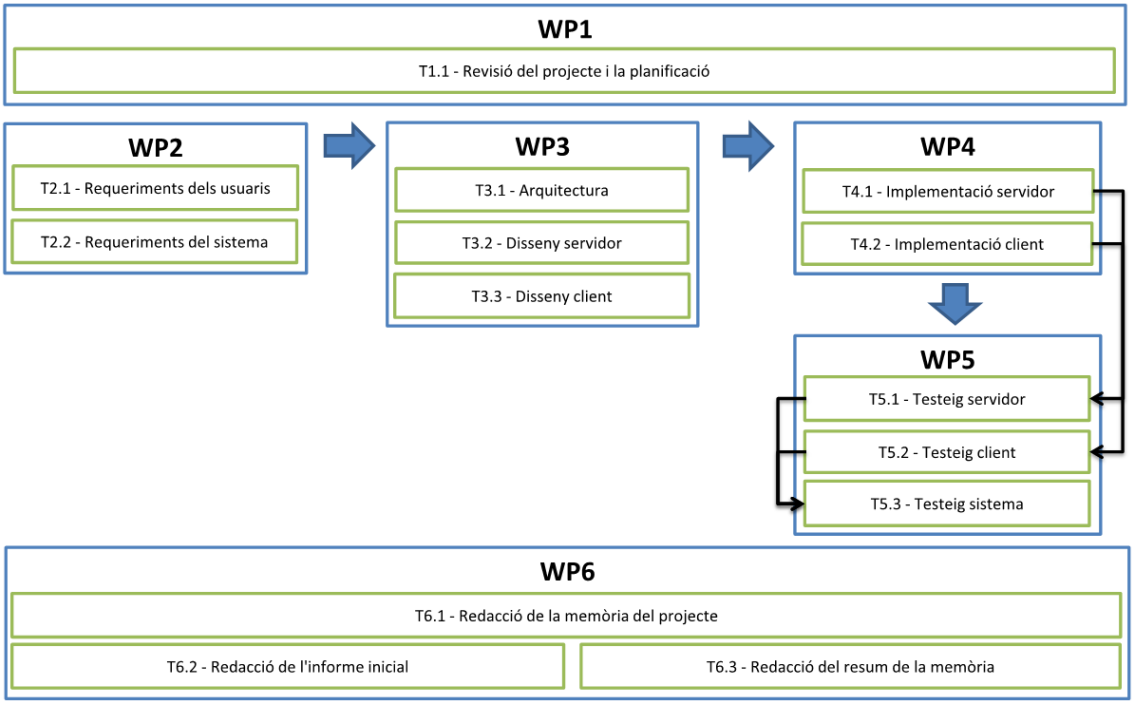

Figure 2.1: Project PERT diagram

Regarding the other WPs, they will be done in parallel to the previous ones, but they will follow the established sequence. **WP2** will give a more clear idea of what is necessary for the project. Once the idea is clear, **WP3** will represent which elements and processes are going to be part of the system and **WP4** will code them in order to create the system functionality. Parallel to WP4, **WP5** will test each functionality as the different parts are ready. In this way, the integration testing will be easier because the possible problems should have been found during development.

2.3 Gantt diagram

Gantt diagram is a graphical representation of the amount of time that each task of the project will take.

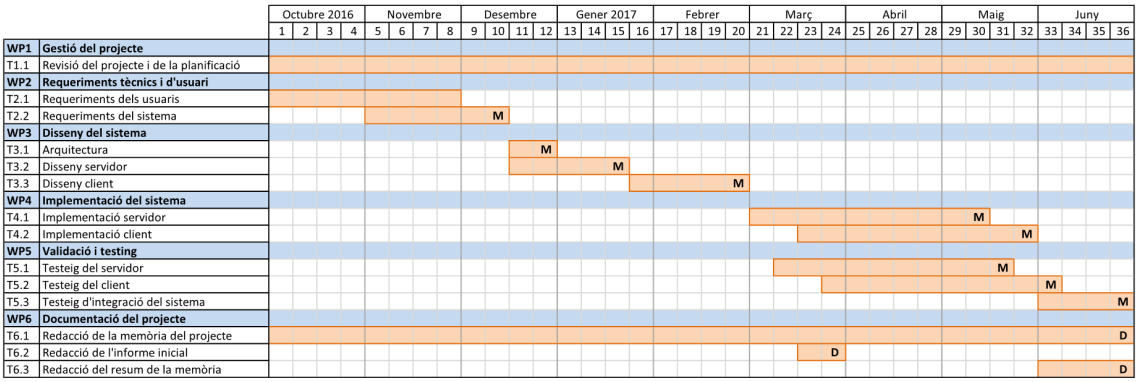

Figure 2.2: Gantt diagram of the project

The project has a duration of **9 months** (36 weeks), beginning on October 2016 and ending on June 2017. Even though a FDP should be done in 4 months, the fact of depending on the availability of users to get the requirements is what has increased the amount of time needed to do the project.

The time assigned to each WP varies depending on the estimated work load of them. So, WPs 2 and 3 will last around **10 weeks** each and they are dependent one from the other (first comes the requirements gathering and, after that, the designing). However, WPs 4 and 5 last **16 weeks** and they are done in parallel, as shown by PERT diagram, because the code will be tested as it's being developed. Even so, having in mind the users' availability problem, the time may be redistributed depending on the remaining time after the requirement gathering.

The diagram also shows the *milestones* (**M**) indicating the estimated time in which a task will be ready and the *deliverables* (**D**) that must be submitted. The different milestones and deliverables of the project are shown below.

2.3.1 Deliverables

Following table shows which are the deliverables to submit during the project and the estimated week in which they will be submitted.

Number	WP	Task	Description	Week
1	6	6.2	Initial report submission	24
2	6	6.1	FDP document submission	36
3	6	6.3	FDP abstract submission	36

Table 2.1: Deliverables table

2.3.2 Milestones

Following table shows which are the milestones to achieve during the project and the estimated week in which they will be achieved.

Number	WP	Task	Description	Week
1	2		Requirements gathered	10
2	3	3.1	End of architecture design	12
3	3	3.2	End of server side design	15
4	3	3.3	End of client side design	20
5	4	4.1	End of server side development	30
6	5	5.1	End of server side testing	31
7	4	4.2	End of client side development	32
8	5	5.2	End of client side testing	33
9	5	5.3	End of integration testing	36

Table 2.2: Milestones table

2.4 Work methodology

This section describes the methodology that will be used for developing the system. The use of this methodology allows to get an extensible and bug-free system in a relative short time.

2.4.1 SOLID principles

SOLID principles [9][10] were described by Robert Cecil Martin [11] around the year 2000. SOLID is an acronym used to describe five basic principles which allow to get a clean, extensible and easy to maintain code. These principles are:

- **Single responsibility principle (SRP):** a class should be only used for a concrete task.
- **Open-closed principle (OCP):** components must be opened to extensions and closed for modifications.
- **Liskov substitution principle (LSP):** a subclass should be able to substitute a superclass without affecting the normal behaviour of the system.
- **Interface segregation principle (ISP):** it is better having different specific interfaces rather than having a general one.
- **Dependency inversion principle (DIP):** classes should depend on abstractions and not on concrete classes.

3 Requirements

Hospital Clínic i Provincial de Barcelona needs a system to allow their physicians to receive different alarms when some kind of risk is present in the patient status and, consequently, react faster to it. The services where the system will be applied are Internal Medicine (IM) and Emergency (ER).

3.1 User requirements

Different requirements gathered from users have been taken into account when the system has been developed. They have been classified as follows:

- System
- Alarms
- Client application
- Internal Medicine
- Emergency

An identifier has been used to distinguish the different requirements. The format of the identifier is this: **REQ-[Classification code]-[Numeric code]**, where:

- **Classification code:** classification where a requirement belongs
- **Numeric code:** differentiates each requirement from the same classification

3.1.1 System requirements (SYS)

Requirement ID	Description
REQ-SYS-01	The alarm must arrive as soon as possible to the physician
REQ-SYS-02	The arrival of the alarm must be guaranteed
REQ-SYS-03	Data privacy must not be compromised at any time
REQ-SYS-04	Users must exist on HIS authenticating them via LDAP

Table 3.1: System requirements (SYS)

3.1.2 Alarm requirements (ALR)

Requirement ID	Description
REQ-ALR-01	The alarms should indicate if they have been relevant or not
REQ-ALR-02	The alarms must contain which parameters have generated them
REQ-ALR-03	The alarms must contain who has seen them

Table 3.2: Alarm requirements (ALR)

3.1.3 Client application requirements (APP)

Requirement ID	Description
REQ-APP-01	The application must be easy to use
REQ-APP-02	The interface has to be user-friendly

Table 3.3: Client application requirements (APP)

3.1.4 Internal Medicine (IM) requirements

Requirement ID	Description
REQ-MI-01	Some alarms have to compare previous and current data
REQ-MI-02	Some alarms have to check medicines administered to a patient

Table 3.4: Internal Medicine (IM) requirements

Alarms defined

Alarm 1: Sepsis

Generated if 3 of the conditions are given. In case of one of them is hypotension, only two conditions are needed.

Parameter	Value
Systolic blood pressure	≤ 100
Breathing frequency	≥ 22
Temperature	≤ 36 o ≥ 38
Heart rate	≥ 100
Oxygen saturation	≤ 90
Glasgow Coma Scale	≤ 14
Lactate	≥ 18
Leukocyte	≤ 4000 o ≥ 12000
Platelet	≤ 150000
Arterial Po2 $\div$ Oxygen %	≤ 400
Bilirubin	≥ 1.2
Creatinine	≥ 1.2
Blood culture	Positive
Urine culture	Positive

Table 3.5: IM alarm 1

Alarm 2: Hypotension and medicine

Generated if both conditions are given.

Parameter	Value
Systolic blood pressure	≤ 120
Antihypertensive or diuretic medicine	Positive

Table 3.6: IM alarm 2

Alarm 3: Heart failure

Generated if 3 conditions are given. If diuresis and daily balance conditions are given, 4 conditions will be necessary.

Parameter	Value	Comments
Heart rate	≥ 100	
Breathing frequency	≥ 30	
Oxygen saturation	≤ 93	
O2 Fio2	$>$ previous value	
O2 Q	$>$ previous value	
Diuresis	≤ 400 ml/day	from 8:00 to 8:00
Daily balance	≥ 1 litre	in 2 consecutive days
Sodium	≤ 135	
Creatinine	≥ 1.3	
Haemoglobin	≤ 90	
Systolic blood pressure	≤ 150	

Table 3.7: IM alarm 3

Alarm 4: High creatinine and medicine

Generated if both conditions are given.

Parameter	Values
Ieca or Arall or Aines or Aminoglucòsids medicine	Positive
Creatinine	≥ 1.3

Table 3.8: IM alarm 4

Alarm 5: Positive culture

Generated if, at least, one of the conditions is given.

Parameter	Value
Urine culture	Positive
Blood culture	Positive

Table 3.9: IM alarm 5

Alarm 6: Hyperglycemia or hypoglycemia

Generated if one of the conditions is given.

Parameter	Value
Glycemia	$\geq$ (200 + sum of the insulin of last 24 hours)
Glycemia	$\leq$ (100 + sum of the insulin of last 24 hours)

Table 3.10: IM alarm 6

3.1.5 Emergency (ER) requirements

Requirement ID	Description
REQ-UR-01	It should be possible to automatically resend a generated alarm if no one has seen it in a certain time period

Table 3.11: Emergency (ER) requirements

Alarms defined

Alarm 1: Medical test validated

Generated if the condition is given.

Parameter	Value	Comments
External and Image Centre Diagnose test	Validated	Resend if it has not been seen after 5 minutes

Table 3.12: ER alarm 1

Alarm 2: Altered gasometry

Generated if one or more conditions are given.

Parameter	Value	Comments
PH	≤ 7.3	
PO2	≤ 60	
PCO2	≥ 50	valid if PH ≤ 7.3

Table 3.13: ER alarm 2

Alarm 3: Renal insufficiency

Generated if one of the conditions is given.

Parameter	Value	Comments
Glomerular filtration calculated (GFC)	≤ 90	If there is no previous GFC
Glomerular filtration calculated (GFC)	$\leq$ previous	If there is a previous GFC

Table 3.14: ER alarm 3

Alarm 4: Ionogram

Generated if one or more conditions are given.

Parameter	Value
Potassium	≤ 3.5 o ≥ 5.5
Sodium	≤ 120

Table 3.15: ER alarm 4

Alarm 5: Glycemia

Generated if the condition is given.

Parameter	Value
Glycemia	≤ 60 o ≥ 40

Table 3.16: ER alarm 5

Alarm 6: Haemoglobin

Generated if one or more conditions are given.

Parameter	Value
Haemoglobin	≤ 9
Hematocrit	≤ 25

Table 3.17: ER alarm 6

Alarm 7: Insufficient sample

Generated if the condition is given.

Parameter	Value	Comments
Laboratory report parameter	= Insufficient sample	Validated or prior to validate document

Table 3.18: ER alarm 7

3.2 Technical requirements

This section describes the software requirements of the project. They have been classified as follows:

- Client application
- Server
- Database
- Software

3.2.1 Client application requirements (CLI)

Requirement ID	Description
REQ-CLI-01	The application should run on mobile platforms
REQ-CLI-02	The application should run on web platforms
REQ-CLI-03	Mobile platform application must receive push notifications

Table 3.19: Client application requirements (CLI)

3.2.2 Server requirements (SER)

Requirement ID	Description
REQ-SER-01	Low resource consumption
REQ-SER-02	Allow creating Representational State Transfer (REST) Web Services (WS)
REQ-SER-03	Allow configuring Hypertext Transfer Protocol Secure (HTTPS) access
REQ-SER-04	Easy to configure
REQ-SER-05	Must be able to send push notifications

Table 3.20: Server requirements

3.2.3 Database requirements (DB)

Requirement ID	Description
REQ-DB-01	Accepting medium-high number of requests
REQ-DB-02	It has to be fast responding to requests
REQ-DB-03	It has to guarantee that data is maintained

Table 3.21: Database requirements

3.2.4 Software requirements (SOF)

Requirement ID	Description
REQ-SOF-01	Free to use (open-source, free versions...)
REQ-SOF-02	It has to be reliable (error-free, stable...)
REQ-SOF-03	It has to be light (low disk storage consumption)

Table 3.22: Software requirements (SOF)

3.3 Decisions

3.3.1 Client

The chosen software for implementing client applications has been Android (mobile application) and HTML, CSS and JS (web application).

Android [12]

Android is an operative system based on Linux designed for mobile devices like smart-phones and tablets, although nowadays it's being popular in wearable devices. Nowadays Google is developing it and maintaining it and has become one of the most used system around the world. Its interface is based on the interaction between the user and the device screen, which makes it easy to use.

HTML, CSS i JS

The use of these technologies has become some kind of standard when creating web applications. The details of each one are described below:

- **HyperText Markup Language (HTML):** it's the standard language for creating web pages. As it is interpreted by the web browser instead of the server, it's possible to view web pages that are stored locally (it's not necessary to connect to a web server).
- **Cascading Style Sheet (CSS):** it's a styling language used to define the appearance a markup language document will have. It was designed to split the content of a document from its presentation, making it easier to be interpreted by browsers, to change its styling and to reuse the same style across different documents.
- **JavaScript (JS):** it's an interpreted programming language (executed using an interpreter, not a compiler). This interpreter is included in almost every web browser, that is, can be executed locally without the need of a dedicated server, as it happens with HTML. Not needing a compiler makes it flexible when developing web applications because it doesn't need neither too many resources nor too many time for being tested.

3.3.2 Server

The chosen software for implementing the server has been Apache Tomcat.

Apache Tomcat [13]

Apache Tomcat is an open-source software which implements Java technologies for servers (servlets, websockets...). Allows creating servers without needing too many resources and with a very acceptable performance.

Similar software:

- **Glassfish:** it's an open-source server more oriented on enterprises as it offers more functionalities than Tomcat. The main inconvenient is that it consumes more resources.

- **Jetty:** it's another open-source software for implementing application servers. It offers similar functionalities to Tomcat, but it's more complex to configure.

3.3.3 Database

The chosen software for DB managing has been PostgreSQL.

PostgreSQL [14]

PostgreSQL is an open-source relational DB system. Allows manipulating a high amount of data in short time thanks to its scalability and assuring the stored data integrity.

Similar DB systems:

- **MySQL:** it's another open-source relational DB system. It's faster and lighter than PostgreSQL, but the amount of data and accesses supported is inferior, as well as the offered data integrity. It's commonly used in web applications where the main goal is to have a great access speed to data.
- **MongoDB:** it's an open-source system too, but it does not use SQL. It's a document oriented system (no tables are used) and it offers fast data access, but data integrity is not guaranteed. It's commonly used in big-data area.

3.3.4 Push notifications

The chosen solution for sending push notifications has been Firebase Cloud Messaging.

Firebase Cloud Messaging [15]

Firebase is a web application development platform firstly created by Google and being currently maintained as a subsidiary of it. Firebase Cloud Messaging (FCM) is a product which provides connection between server and devices through cloud. It's totally free and provides support for Android, iOS, Web applications, C++ and Unity.

Similar solutions found, like Amazon Simple Notificacion Service or Kinvey, require creating a Firebase project in order to allow sending notifications to devices. As they are not a so straight-forward solution, they have not been considered as an alternative solution.

3.3.5 Software tools

The chosen software for developing the project has been, Eclipse IDE, Astah modelling tool and Bitbucket code repository.

Astah [16]

Astah, previously knew as JUDE, is an Unified Modelling Language (UML) modelling tool created by Japanese company Change Vision. This software allows creating different diagrams (use case, classes, sequence...) for designing the system before implementing it. The free version has been chosen for this FDP, but there is also a paid version (Professional) which offers more functionalities.

Similar modelling tools:

- **Modelio:** it's an open-source software created by ModelioSoft. It offers similar functionalities as Astah, but its use is not too friendly and it presents some stability failures.
- **Papyrus:** this tool is similar to Modelio, as it's open-source and offers different modelling options. The main inconvenient is that is not an independent software, it's a plug-in from another program (Eclipse Modelling Tools) and needs too much storage space for its installation.

Eclipse IDE [17]

Eclipse is a multi-platform Integrated Development Environment (IDE) which offers different tools for Java programming, although there are different plug-ins for programming in many other languages. It's an open-source tool created by Eclipse Foundation community, who offers continuous support for all its solutions.

Android Studio [18]

Android Studio is an IDE focused on Android application programming. Currently is developed by Google and the updates are available continuously. It offers tools for programming applications and an emulator to test these applications without the need of installing them into a real device.

Similar IDEs:

- **NetBeans:** it's an open-source IDE for Java programming. Which makes it different to Eclipse is that it's even more focused on Java and there are less plug-ins for developing other languages.
- **IntelliJ IDEA:** this is one of the most powerful IDE nowadays. It offers a lot of functionalities for Java and Android development. The main inconvenient is that a lot of functionalities are only available in its paid version.

Bitbucket [19]

Bitbucket is an on-line code repository service for projects using Git or Mercurial control version systems. It allows to have all the code stored securely and to make it easily portable to other systems. Free version offers unlimited number of repositories, which can be public or private. Private ones are restricted to be used by, at most, 5 users. If there is need for more users, there are different premium versions, which prices depend on the number of users required.

Similar repository services:

- **GitHub:** it's a Git repository service similar to Bitbucket. The main difference is that the free version only offers public repositories and private ones are only available for paid version.
- **GitLab:** this service offers unlimited public and private repositories, accessible to an unlimited number of users for the free version. The main difference is that GitLab only supports Git system while Bitbucket also supports Mercurial.

4 System design

4.1 Introduction

System design has defined the different parts that will form the system before implementing it. In order to define those parts, an analysis of the requirements and a **use-case** definition has been done. Use-cases allow representing actions that different actors (people or other systems) will do with the system.

An identifier matching the following naming convention has been used in order to differentiate the use-cases: **CU-[System code]-[Scenario code]-[Number code]**, where:

- **System code:** system to which the use-case belongs
- **Scenario code:** scenario to which the use-case belongs
- **Number code:** differentiates use-cases from same system and scenario

Once use-cases have been defined, the different objects from use-cases have been created along with some **UML sequence diagrams**, which represent interactions between those objects. Sequence diagrams are identified similar to use-cases, using this naming convention **SEQ-[System code]-[Scenario code]-[Number code]**.

From sequence diagrams, **UML class diagrams** have been created to define which parameters and methods each object will contain. Finally, the architecture of the Smart Clinical Alarms Service (SCAS) has been created to show how it will be integrated into the Hospital Information System (HIS).

4.2 Database design

Database is used by SCAS server for storing data related to service administration and alarm processing and sending. Figure 4.1 represents the relational model followed by DB, where primary keys are underlined and foreign keys are marked with # symbol.

User table makes reference to physicians and system administrator. They are defined by their user name in the HIS, a device token, a role and a field to indicate if they are on their work shift. The token is used to identify to which device will the push notification be sent, and the role will say whether the user is an administrator or not.

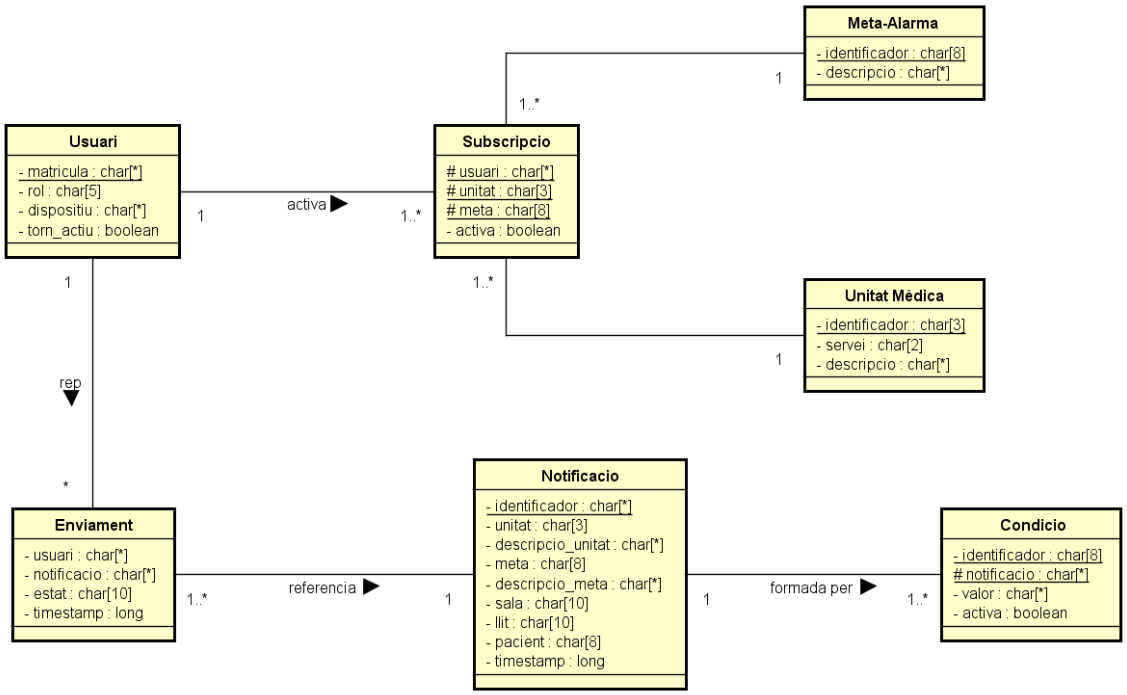

Figure 4.1: DB relational model

A **medical unit** makes reference to a concrete section from a hospital service (for example, thoracic pain unit from emergency service) from where the patient data will be analysed. It is defined by an identifier, the service to which the unit belongs and a brief description. A **meta-alarm** is the definition of which conditions have to be presented in patient data in order to raise an alarm. Into DB there are only stored the meta-alarm identifier and its description.

To allow the users to decide which alarms they want to receive and which not, a **subscription** that joins a meta-alarm and a medical unit is used. Every user will have a subscription to every unit and meta-alarm combination, but they will choose which ones they want to activate or deactivate.

Once the server checks if the patient data shows some risk, the system will generate a **notification** containing an identifier, the meta-alarm that has been used to determinate the risk, the medical unit, the room and the bed where the data has been taken from and the creation timestamp of the alarm. Along with the notification, the **conditions** matching the meta-alarm criteria will be included, so the physician can know more directly why the alarm has been triggered.

For each notification there is, at least, a **message** associating the user who will receive the notification and the notification that has been sent. The message will also have the timestamp

when the user has received the alarm and a status that will help to know if the user has indicated, for example, that the alarm is a false positive.

Following figure shows a UML state diagram representing the different states of a message:

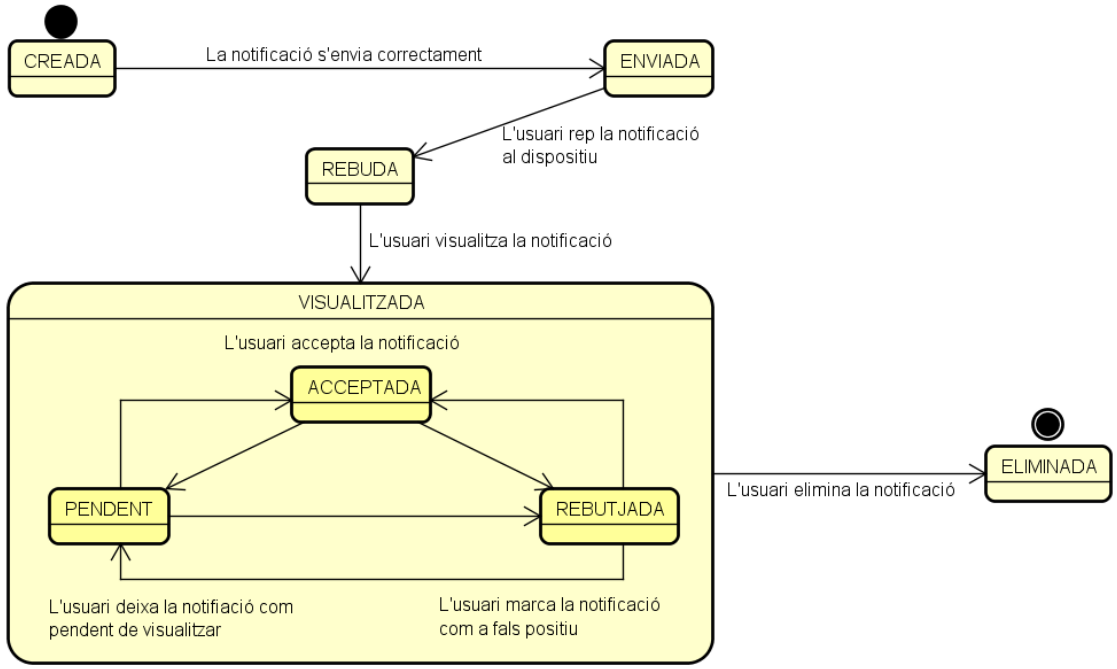

Figure 4.2: Notifications state diagram

The service administrative part is related to the user, subscription and notifications management. The actions allowed by the DB should be the following:

Table	Create	Read	Update	Delete
Users	x	x	x	x
Notifications		x		
Message		x	x	
Subscriptions		x	x	

Table 4.1: Administrative actions by the server over the DB

On the other side, the processing part of the service is in charge of reading patient data, process them to evaluate if there is a risk and send the alarms to the users if needed. The actions allowed by the DB should be the following:

Table	Create	Read	Update	Delete
Notifications	x			
Message	x			
Subscriptions		x		

Table 4.2: Processing actions by the server over the DB

4.3 Server design (administration)

Server administrative part allows clients to manage DB information related to users, subscriptions and notifications. Server communication is done via Representational State Transfer (REST) Web Services (WS).

4.3.1 Use-case description

Every use-case from the server include the following use-cases related to SQL queries:

SQL query request (CU-SER-SQL-01)

CU-SER-SQL-01	*SQL query request*
Actors	User, notification and subscription services
Description	The service performs a query to get data from DB
Main scenario	*Successful query*
Step 1	The service connects to the DB using a **controller**
Step 2	The query is sent to the DB
Step 3	The service gets the data
Step 4	The service disconnects from DB
Step 5 (Final)	The service generates a response containing the data
Variation A	*Connection failure*
Step 1	The service can not connect to the DB
Step 2 (Final)	The service generates an error response
Variation B	*No data found*
Step 3	The service doesn't get data
Step 4	The service disconnects from DB
Step 5 (Final)	The service generates a response without any data
Variation C	*Disconnection failure*
Step 4	The service can not disconnect from DB

Step 5 (Final)	The service generates an error response

Table 4.3: CU-SER-SQL-01 description

SQL modification request (CU-SER-SQL-02)

CU-SER-SQL-02	*SQL modification request*
Actors	User, notification and subscription services
Description	The service performs a query to create, update or delete data from DB
Post-conditions	
Post-condition 1	If there is no error, the DB content will change
Post-condition 2	If there is an error, the DB content won't change
Main scenario	*Successful modification*
Step 1	The service connects to the DB using a **controller**
Step 2	The query is sent to the DB
Step 3	The service doesn't receive an error
Step 4	The service disconnects from DB
Step 5 (Final)	The service generates a sucessful response
Variation A	*Connection failure*
Step 1	The service can not connect to the DB
Step 2 (Final)	The service generates an error response
Variation B	*Modification failure*
Step 3	The service receives an error
Step 4	The service disconnects from DB
Step 5 (Final)	The service generates a modification failure response
Variation C	*Disconnection failure*
Step 4	The service can not disconnect from DB
Step 5 (Final)	The service generates an error response

Table 4.4: CU-SER-SQL-02 description

Authentication (AUT)

Scenario from the operations related to user authentication into the SCAS. The UML diagram from the scenario is the following:

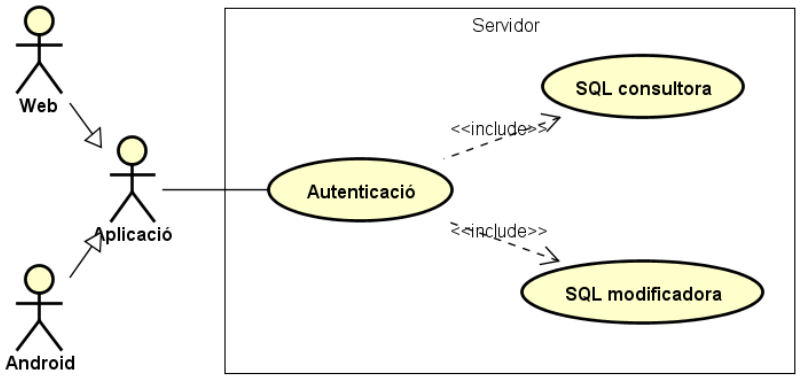

Figure 4.3: "Authentication" scenario UML diagram

User authentication (CU-SER-AUT-01)

CU-SER-AUT-01	*User authentication*
Actors	Web application - Android application
Description	The application requests authenticating a user by giving its user name and password
Main scenario	*User authenticated successfully*
Step 1	The **application** requests the authentication of the user
Step 2	The service checks if the user exists in the HIS via LDAP using the given user name and password
Step 3	The user is successfully authenticated via LDAP
Step 4	The service creates an **SQL query request** to check if the user exists on the SCAS
Step 5	The service performs the request
Step 6	*SQL query request (CU-SER-SQL-01)*
Step 7	The user does not exist in the SCAS
Step 8	The service creates an **SQL modification request** to create the user into the SCAS
Step 9	The service performs the request
Step 10	*SQL modification request (CU-SER-SQL-02)*
Step 11	The user has been created successfully
Step 12	The service generates a session token
Step 13 (Final)	The service sends the response and the token to the application

Variation A	*User exists in SCAS*
Step 7	The user exists in the SCAS
Step 8	The service generates a session token
Step 9 (Final)	The service sends the response and the token to the application
Variation B	*User is not authenticated on LDAP*
Step 3	The user is not successfully authenticated via LDAP
Step 4 (Final)	The service sends a response to the application

Table 4.5: CU-SER-AUT-01 description

Users (USR)

Scenario from the operations related to user data manipulation into DB. The UML diagram from the scenario is the following:

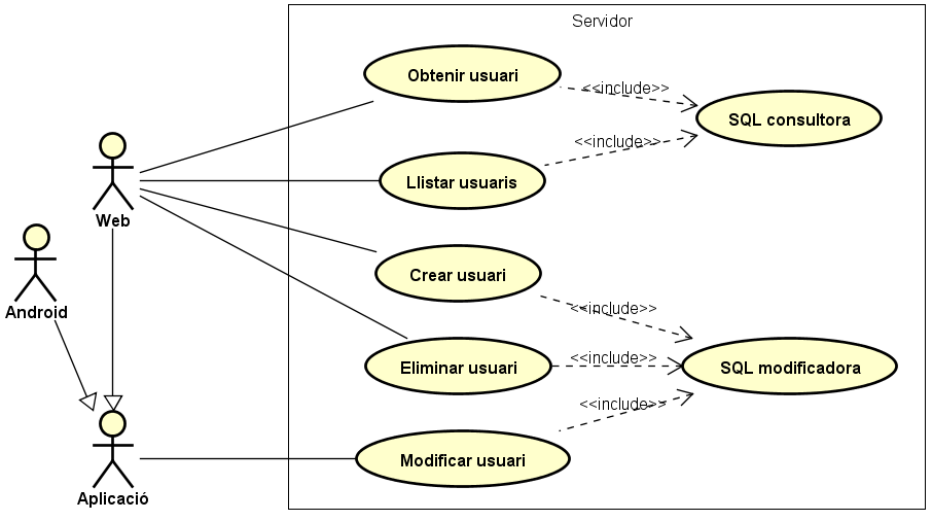

Figure 4.4: "Users" scenario UML diagram

List users (CU-SER-USR-01)

CU-SER-USR-01	*List users*
Actors	Web application
Description	The application requests the non-administrator user list from DB
Main scenario	*Users listed successfully*
Step 1	The **application** requests the user list to **user service**
Step 2	The service creates an **SQL query request** to get the user list
Step 3	The service performs the request

Step 4	*SQL query request (CU-SER-SQL-01)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.6: CU-SER-USR-01 description

Create user (CU-SER-USR-02)

CU-SER-USR-02	*Create user*
Actors	Web application
Description	The application requests the creation of a user into DB by giving its user name
Main scenario	*User created successfully*
Step 1	The **application** requests the creation of a user to the **user service**
Step 2	The service creates an **SQL modification request** to create the user
Step 3	The service performs the request
Step 4	*SQL modification query (CU-SER-SQL-02)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.7: Description del CU-SER-USR-02

Delete user (CU-SER-USR-03)

CU-SER-USR-03	*Delete user*
Actors	Web application
Description	The application requests deleting a user by giving its user name
Main scenario	*User deleted successfully*
Step 1	The **application** requests deleting a user to the **user service**
Step 2	The service creates an **SQL modification request** to delete the user
Step 3	the service performs the request
Step 4	*SQL modification query (CU-SER-SQL-02)*

Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.8: CU-SER-USR-03 description

Update user (CU-SER-USR-04)

CU-SER-USR-04	*Update user*
Actors	Web application - Android application
Description	The application requests the update of a user by giving its user name and the updates to apply
Main scenario	*User updated successfully*
Step 1	The **application** requests updating a user to the **user service**
Step 2	The service creates an **SQL modification request** for updating user data
Step 3	The service performs the request
Step 4	*SQL modification query (CU-SER-SQL-02)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.9: CU-SER-USR-04 description

Get user (CU-SER-USR-05)

CU-SER-USR-05	*Get user*
Actors	Web application
Description	The application requests information from a concrete user from the DB by giving its user name
Main scenario	*User obtained successfully*
Step 1	The **application** requests the user data to the **user service**
Step 2	The service creates an **SQL query request** to get data from a user
Step 3	The service performs the request
Step 4	*SQL query request (CU-SER-SQL-01)*
Step 5	The service gets the response

Step 6 (Final)	The service sends the response back to the application

Table 4.10: CU-SER-USR-05 description

Subscriptions (SUB)

Scenario from the operations related to subscriptions data manipulation into DB. The UML diagram from the scenario is the following:

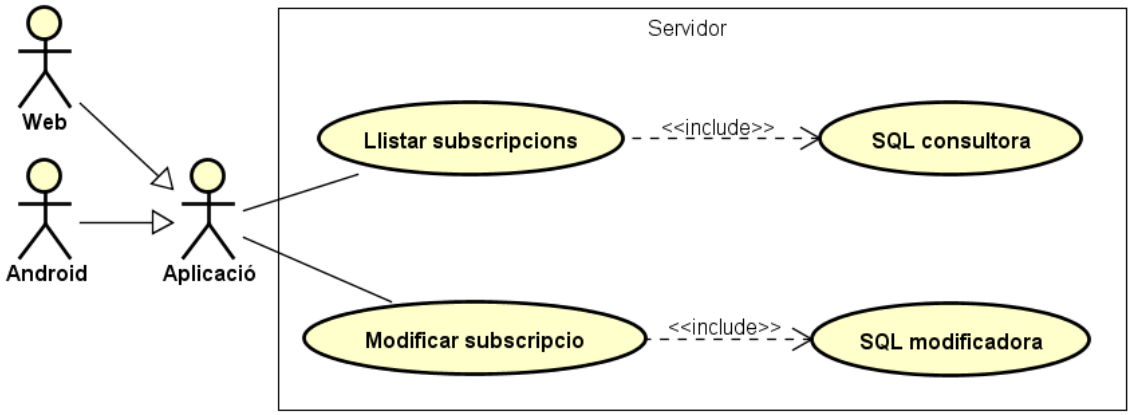

Figure 4.5: "Subscriptions" scenario UML diagram

List subscriptions (CU-SER-SUB-01)

CU-SER-SUB-01	*List subscriptions*
Actors	Web application - Android application
Description	The application requests the list of subscriptions from a concrete user by giving its user name
Main scenario	*Subscriptions listed successfully*
Step 1	The **application** requests the subscriptions to the **subscription service**
Step 2	The service creates an **SQL query request** to get the subscriptions
Step 3	The service performs the request
Step 4	*SQL query request (CU-SER-SQL-01)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.11: CU-SER-SUB-01 description

Update subscriptions (CU-SER-SUB-02)

CU-SER-SUB-02	*Update subscriptions*
Actors	Web application - Android application
Description	The application requests updating the subscriptions from a user by giving its user name, the subscriptions involved and the updates to apply
Main scenario	*Subscriptions updated successfully*
Step 1	The **application** requests updating the subscriptions from a user to the **subscription service**
Step 2	The service creates an **SQL modification request** to update the subscriptions
Step 3	The service performs the request
Step 4	*SQL modification request (CU-SER-SQL-02)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.12: CU-SER-SUB-02 description

Notifications (NOT)

Scenario from the operations related to notifications data manipulation into DB. The UML diagram from the scenario is the following:

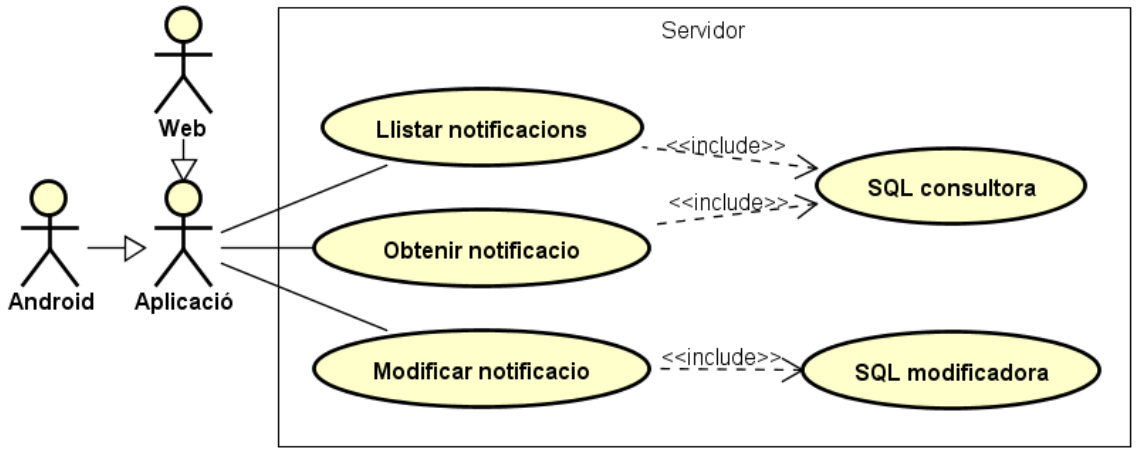

Figure 4.6: "Notifications" scenario UML diagram

List notifications (CU-SER-NOT-01)

CU-SER-NOT-01	*List notifications*
Actors	Web application - Android application
Description	The application requests the notifications list from a user by giving its user name
Main scenario	*Notifications listed successfully*
Step 1	The **application** requests the notifications list from a user to the **notification service**
Step 2	The service creates an SQL query request to get the notifications
Step 3	The service performs the request
Step 4	*SQL query request (CU-SER-SQL-01)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.13: CU-SER-NOT-01 description

Update notification (CU-SER-NOT-02)

CU-SER-NOT-02	*Update notification*
Actors	Web application - Android application
Description	The application requests updating a concrete notification from a user by giving its user name, the notification involved and the updates to apply
Main scenario	*Notifications updated successfully*
Step 1	The **application** requests updating a notification to the **notification service**
Step 2	The service creates an **SQl modification request** to update the notification
Step 3	The service performs the request
Step 4	*SQL modification request (CU-SER-SQL-02)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.14: CU-SER-NOT-02 description

Get notification (CU-SER-NOT-03)

CU-SER-NOT-03	*Get notification*
Actors	Web application - Android application
Description	The application requests data of a concrete notification from a user by giving its user name and the notification identifier
Main scenario	*Notification obtained successfully*
Step 1	The **application** requests getting the specific notification to the **notification service**
Step 2	The service creates an **SQL query request** to get the notification
Step 3	The service performs the request
Step 4	*SQL query request (CU-SER-SQL-01)*
Step 5	The service gets the response
Step 6 (Final)	The service sends the response back to the application

Table 4.15: CU-SER-NOT-03 description

4.3.2 Sequence diagrams

In general terms, the sequence followed by the use-cases related to list and get data is shown in figure 4.7. The service generates a query which uses a controller to connect to the database using the database driver. It performs the query and get the results. Finally, the results are gathered and formatted by the service into an HTTP response that is sent to the requester.

In a similar way, the sequence followed by use-cases related to create, update and delete data is shown in figure 4.8. The steps are almost the same but the response only contains if the modifications into the database have been successful.

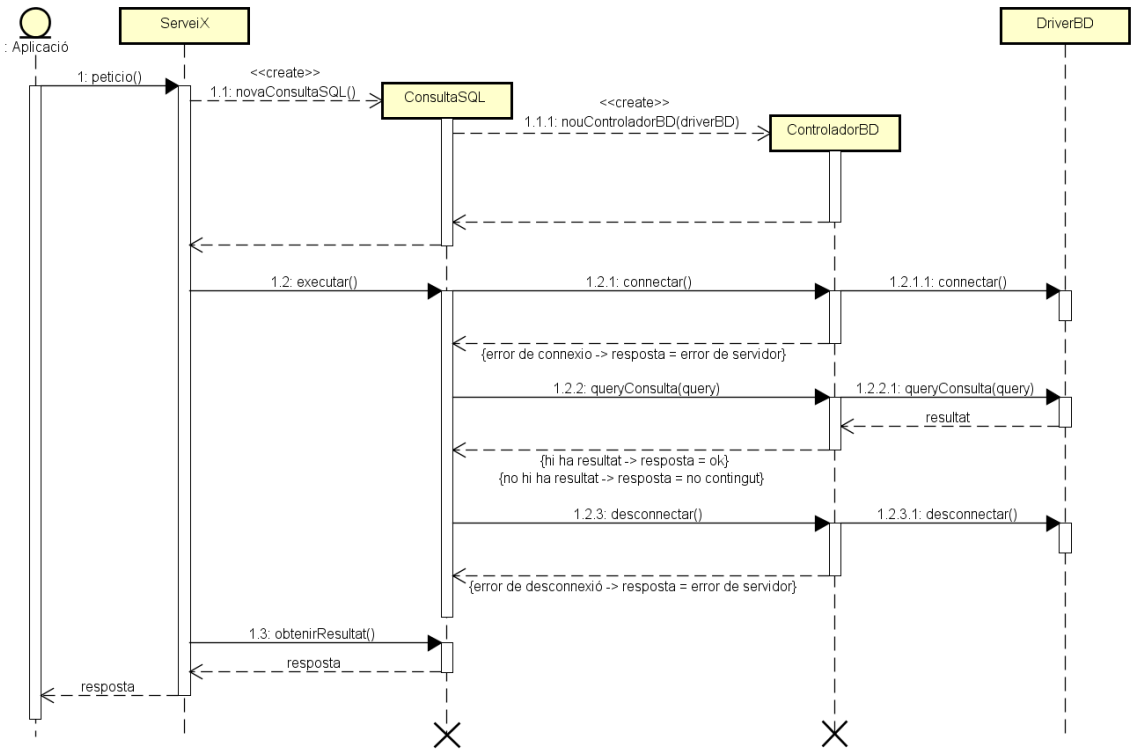

Figure 4.7: Query requests sequence diagram

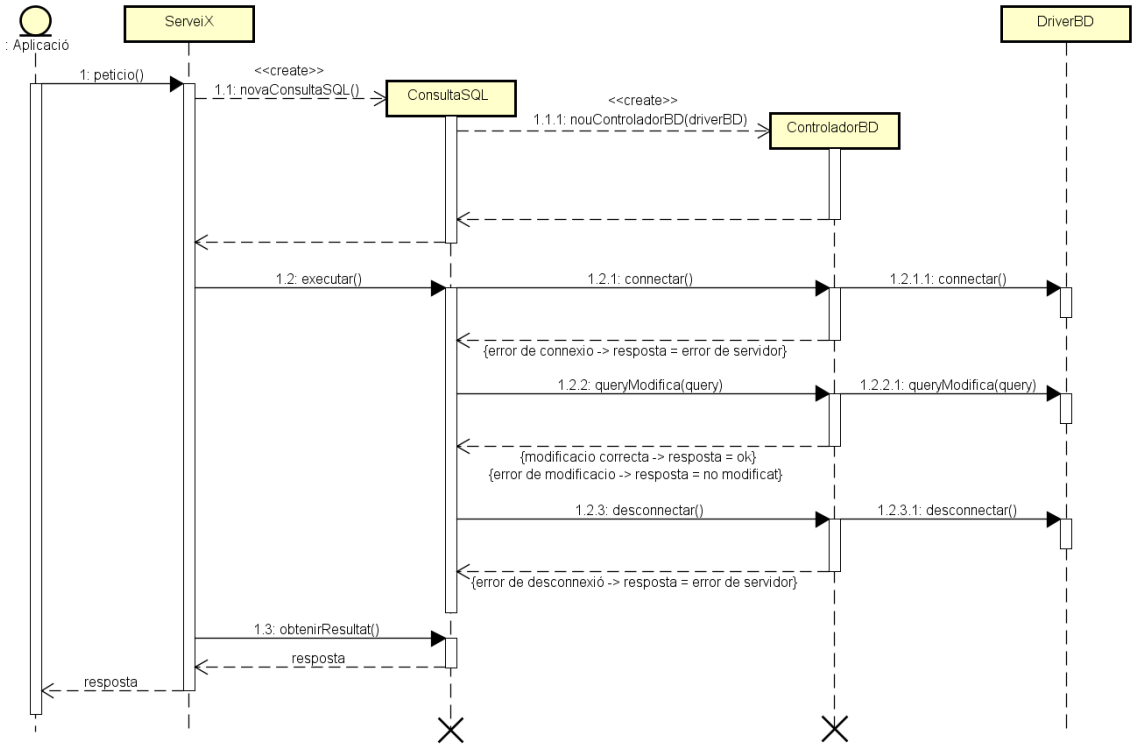

Figure 4.8: Modification requests sequence diagram

4.3.3 Class diagram

The diagram of the components composing the server is the following:

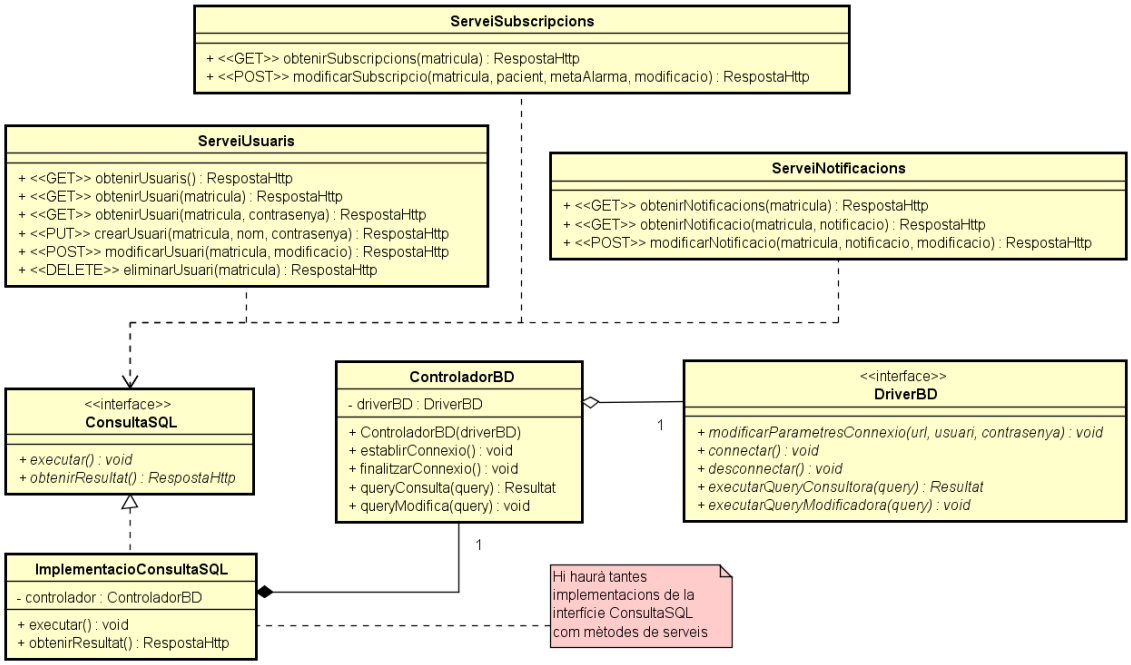

Figure 4.9: Server class diagram

Service classes (ServeiUsuaris, ServeiSubscripcions, ServeiNotificacions) are in charge of publish the necessary methods for managing the different operations (create, update, read and delete) that can be done into the DB

The SQL **queries** are defined by the ConsultaSQL interface. Each query requested to the DB is done using a class (ImplementacioConsultaSQL) which implements this interface. In this way, each class will have a single responsibility instead of grouping a set of queries into a single class.

Each query includes a DB **controller** (ControladorBD) which is in charge of sending the queries to the DB server and returning the response. To send the query, the controller uses a **driver** (which implements the DriverBD interface) that creates the connection to the DB and executes the queries over the data.

4.4 Server design (processing)

Processing part of the server is in charge of getting patient data, determinate if they generate an alarm and, if they do, send the notifications to the users. The processing will be initiated by the SCAS or by the HIS instead of the user.

4.4.1 Use-case description

Processing (PRO)

Scenario related to HIS's data processing. Also the actions related to create and send notifications to users are included. The UML diagram from the scenario is the following:

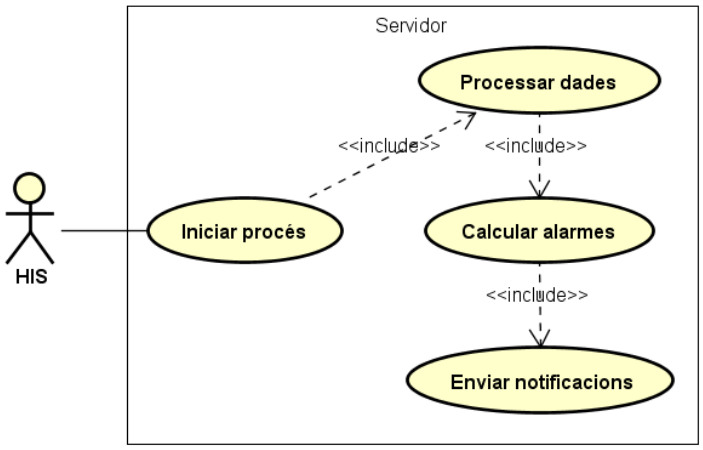

Figure 4.10: "Processing" scenario UML diagram

Start processing (CU-SER-PRO-01)

CU-SER-PRO-01	*Start processing*
Actors	HIS
Description	The HIS tells SCAS that patient data have been updated
Main scenario	*Process started successfully*
Step 1	The HIS requests start data processing to **processing service**
Step 2	The service starts processing data asynchronously (*CU-SER-PRO-02*)
Step 3 (Final)	The service returns a successful response to the request

Table 4.16: CU-SER-PRO-01 description

The sequence diagram related to the use-case is the following:

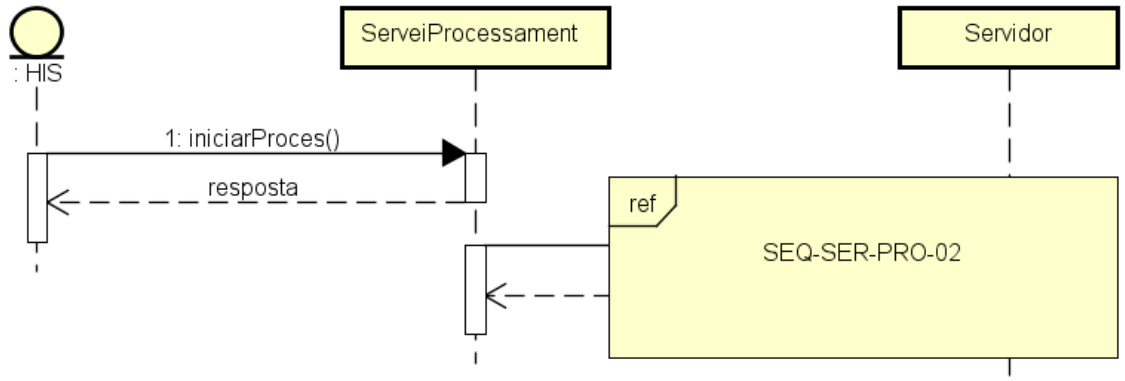

Figure 4.11: SEQ-SER-PRO-01 diagram

Data processing (CU-SER-PRO-02)

CU-SER-PRO-02	*Data processing*
Actors	Server
Description	The server gets patient data from HIS to check later if there is some kind of alarm
Pre-conditions	
Pre-condition 1	The data processing has been requested (*CU-SER-PRO-01*)
Main scenario	*Data processed successfully*
Step 1	The server requests patient data to HIS
Step 2	The server gets patient data
Step 3	The server organises the data
Step 4 (Final)	The server initiates the alarm calculation (*CU-SER-PRO-03*)

Table 4.17: CU-SER-PRO-02 description

The sequence diagram related to the use-case is the following:

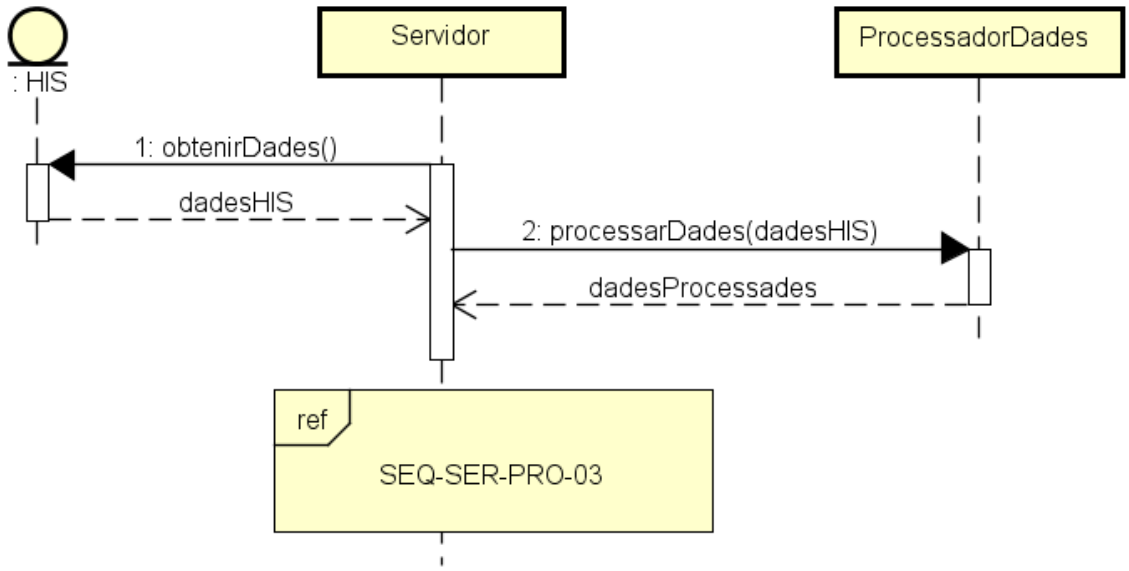

Figure 4.12: SEQ-SER-PRO-02 diagram

Alarm calculation (CU-SER-PRO-03)

CU-SER-PRO-03	*Alarm calculation*
Actors	Server
Description	The server starts calculating the alarms using the obtained data
Pre-conditions	
Pre-condition 1	The HIS data has been processed (*CU-SER-PRO-02*)
Main scenario	*Alarms calculated successfully*
Step 1	The server tells the calculator which data is going to be checked
Step 2	The server tells the calculator which meta-alarms are going to be checked
Step 3	The server tells the calculator to start calculating
Step 4	The calculator checks each data with each meta-alarm
Step 5	The calculator detects that an alarm is generated
Step 6 (Final)	The calculator requests the server to generate a notification with the generated alarm (*CU-SER-PRO-04*)
Variation A	*No alarms have been generated*
Step 5 (Final)	The calculator doesn't detect any alarm

Table 4.18: CU-SER-PRO-03 description

The sequence diagram related to the use-case is the following:

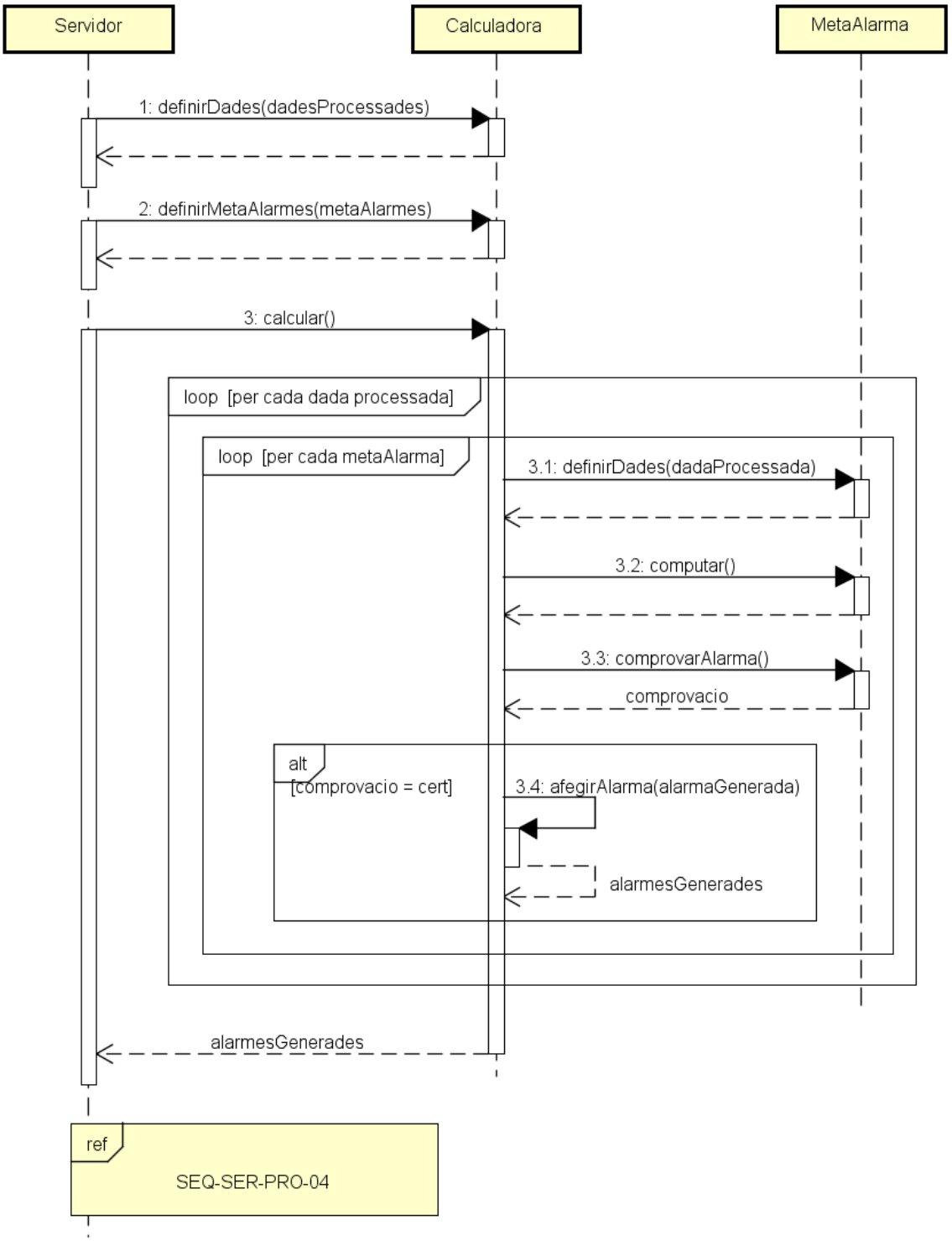

Figure 4.13: SEQ-SER-PRO-03 diagram

Send notifications (CU-SER-PRO-04)

CU-SER-PRO-04	*Send notifications*
Actors	Server
Description	The server sends notifications containing the generated alarm information
Pre-conditions	
Pre-condition 1	The calculator has generated an alarm (*CU-SER-PRO-03*)
Main scenario	*Notifications sent successfully*
Step 1	The server receives the generated alarm
Step 2	The server creates an SQL query request to get the list of active subscriptions for the alarm
Step 3	The server performs the query (*CU-SER-SQL-01*)
Step 4	The server gets the subscriptions list
Step 5	The server creates an SQL modification request to create the notification and as many messages as subscriptions into the DB
Step 6	The server performs the request (*CU-SER-SQL-02*)
Step 7	The server creates an SQL query request to get the device token from the users to whom the notification has to be sent
Step 8	The server performs the request (*CU-SER-SQL-01*)
Step 9	The server gets the token list
Step 10	The server encrypts the alarm information
Step 11 (Final)	For each token, the server sends the push notifications through Firebase Cloud Messaging
Variation A	*No subscriptions have been obtained*
Step 4 (Final)	The server gets no subscriptions
Variation B	*Error while creating notifications and messages*
Step 7 (Final)	The notification and/or the messages have not been created into DB
Variation C	*No tokens have been obtained*
Step 9 (Final)	The server gets no tokens

Table 4.19: CU-SER-PRO-04 description

The sequence diagram related to the use-case is the following:

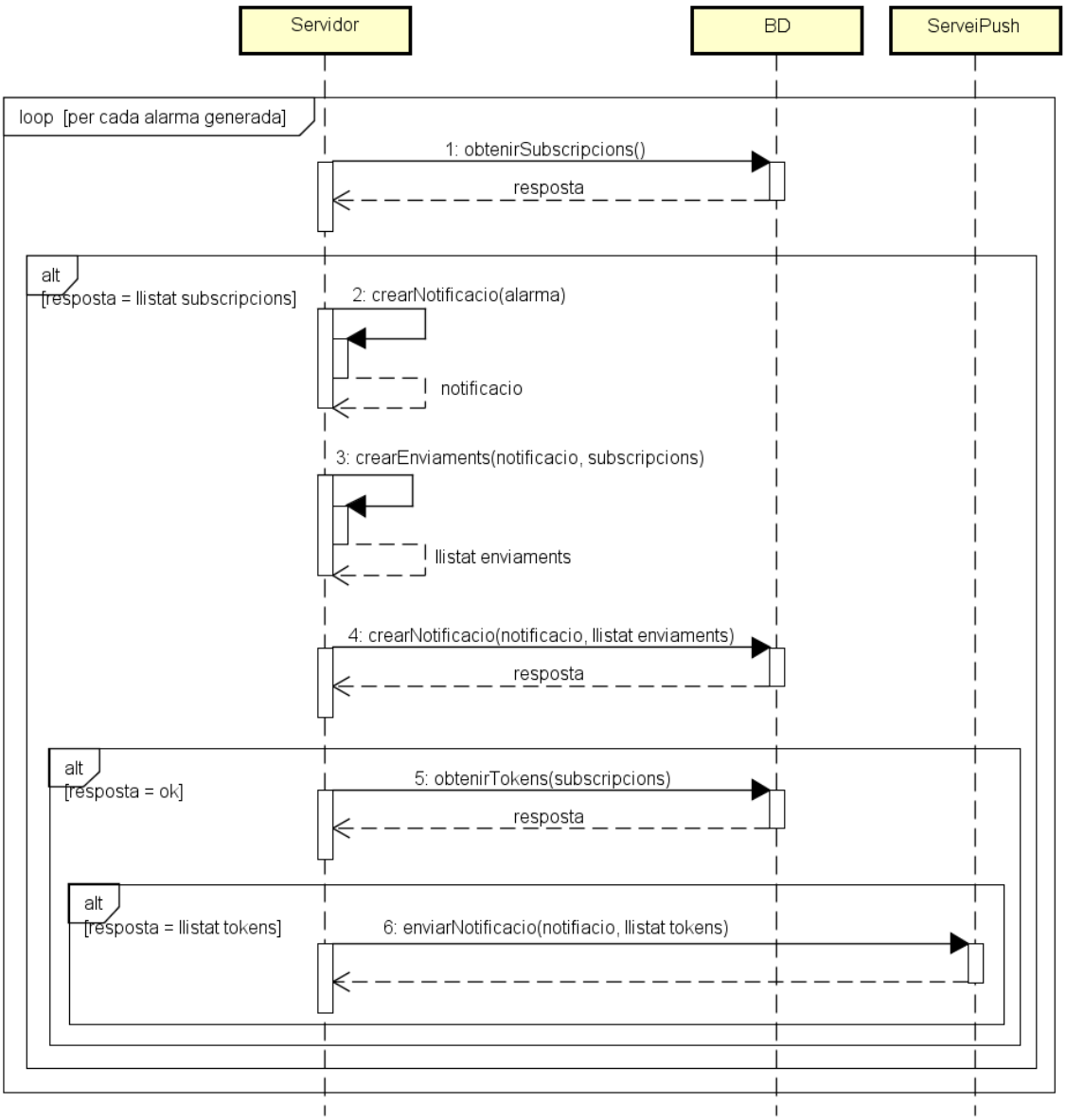

Figure 4.14: SEQ-SER-PRO-04 diagram

4.4.2 Class diagram

The diagram of the components composing the server processing part is the following:

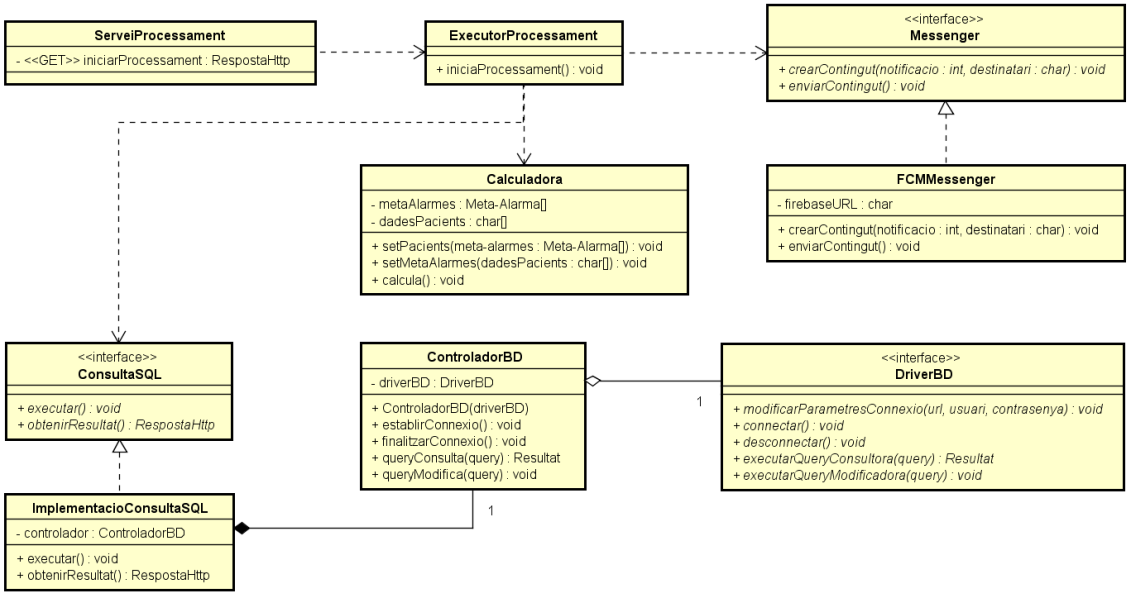

Figure 4.15: Server UML class diagram

In this case, the **service** class (ServeiProcessament) publishes the method needed to let the server know that the processing should be initiated. Then the ExecutorProcessament class will act as an **orchestrator** to get data from HIS, calculate the alarms (Calculadora), perform the DB queries and send the **push notifications** (FCMMessenger) to the involved users.

4.5 Client application design

Client application allows the user to interact with system data and, using the mobile version, receive push notifications generated by the service. To access the application, the user needs to authenticate using her user name and password. Once the user has been authenticated, she will be able to navigate to different sections depending on her role. In this way, users (with user or administrator role) can manage theirs subscriptions and notifications, but only administrator users are allowed to manage the system users.

Figure 4.16 shows a wireframe representing the approximate design the Android application will have and how the user will navigate through the different sections of it.

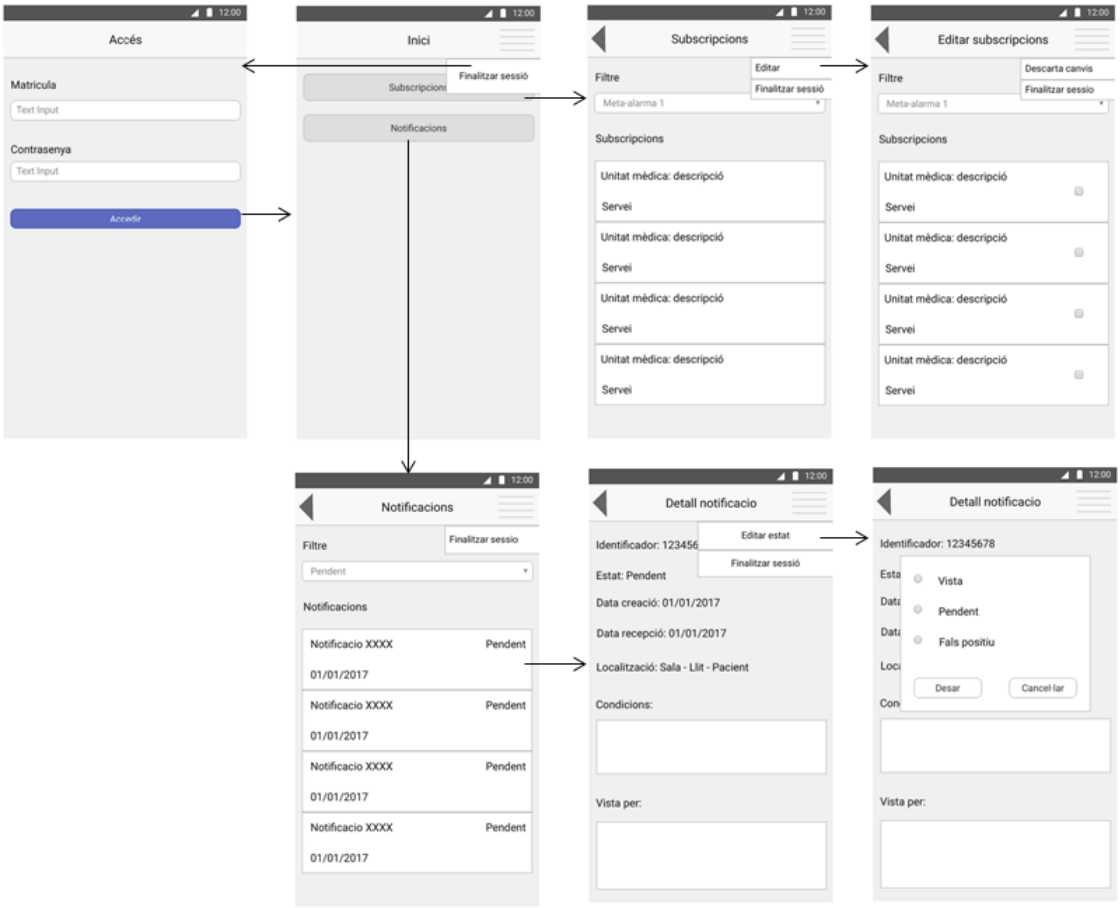

Figure 4.16: Android app wireframes

4.5.1 Use-case description

Authentication (AUT)

Scenario related to how the user is authenticated in order to access the application. The UML diagram from the scenario is the following:

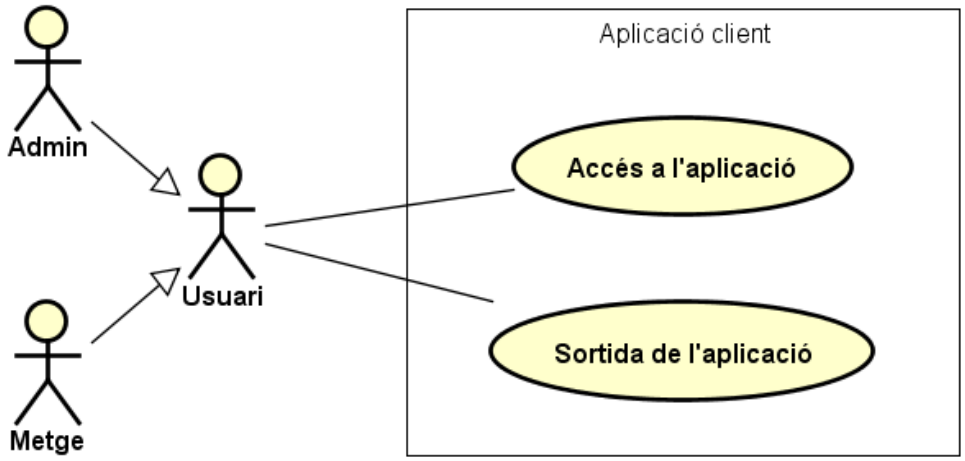

Figure 4.17: "Authentication" scenario UML scenario

Application log-in (CU-APP-AUT-01)

CU-APP-AUT-01	*Application log-in*
Actors	Physician - Admin
Description	The user wants to access to the application. Introduces user name and password into the form and presses the log-in button
Pre-conditions	
Pre-condition 1	The application has not started a user session
Post-conditions	
Post-condition 1	If the access has been successful, the application starts a user session
Post-condition 2	If not, the application doesn't start a user session
Main scenario	*Successful access*
Step 1	User name and password are sent to the **server** (*CU-SER-USR-05*)
Step 2	The server returns the user information
Step 3	The **application** starts a **session** with the user data
Step 4 (Final)	The application sends the user to the main screen
Exception A	*Authentication failure*
Step 2	The server doesn't return user data
Step 3 (Final)	The application tells the user that introduced credentials are wrong
Exception B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3 (Final)	The application informs the user about the error

Table 4.20: CU-APP-AUT-01 description

The sequence diagram related to the use-case is the following:

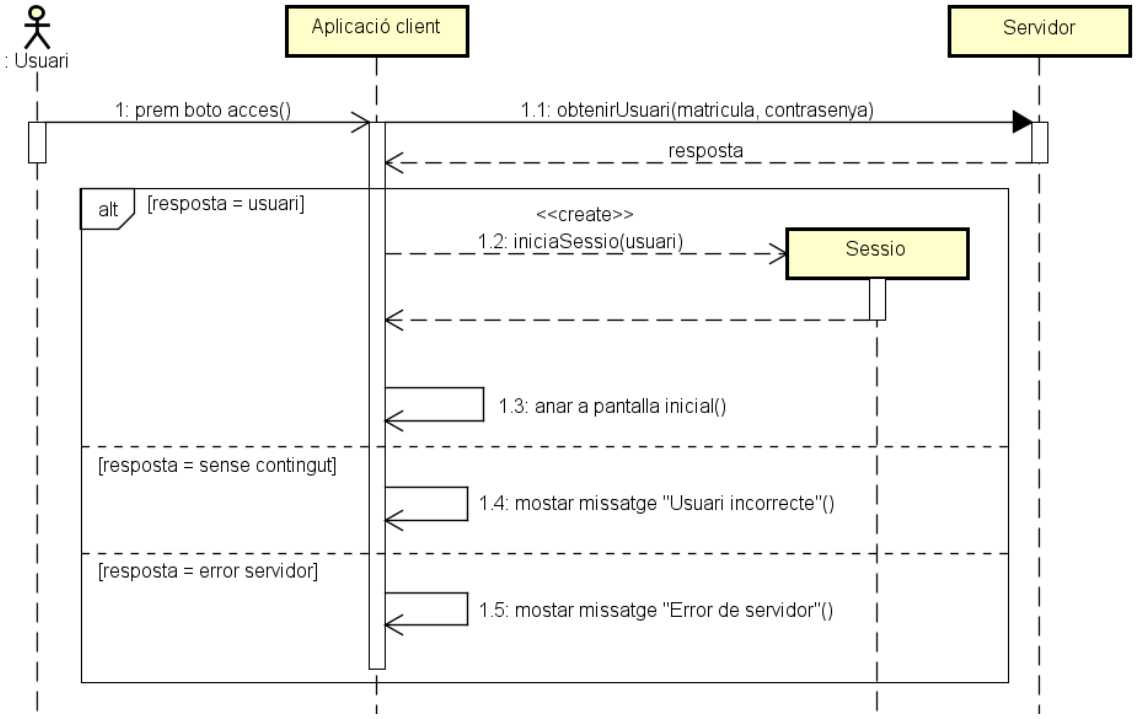

Figure 4.18: SEQ-APP-AUT-01 diagram

Application log-out (CU-APP-AUT-02)

CU-APP-AUT-02	*Application log-out*
Actors	Physician - Admin
Description	The user wants to log-out from the application. Presses the Log-out button placed in the upper right corner of the screen.
Pre-conditions	
Pre-condition 1	The application has started a user session
Post-conditions	
Post-condition 1	If the user confirms, the application will finish the user session
Post-condition 2	If the user doesn't confirm, the session will remain started
Main scenario	*Successful log-out*
Step 1	The **application** pops a confirmation dialog
Step 2	The user confirms the application log-out
Step 3	The application finishes the **session**
Step 4 (Final)	The application sends the user to the log-in screen
Exception A	*The user doesn't confirm*
Step 2	The user doesn't accept the log-out

Step 3 (Final)	The application keeps the session and remains in the current screen

Table 4.21: CU-APP-AUT-02 description

The sequence diagram related to the use-case is the following:

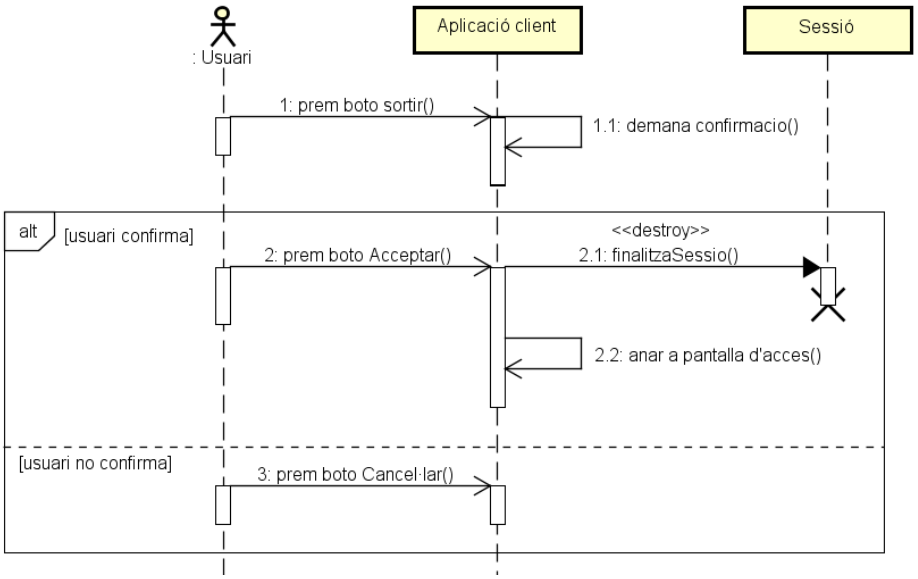

Figure 4.19: SEQ-APP-AUT-02 diagram

Subscriptions (SUB)

Scenario related to the subscription management done by the user into the application. The UML diagram from the scenario is the following:

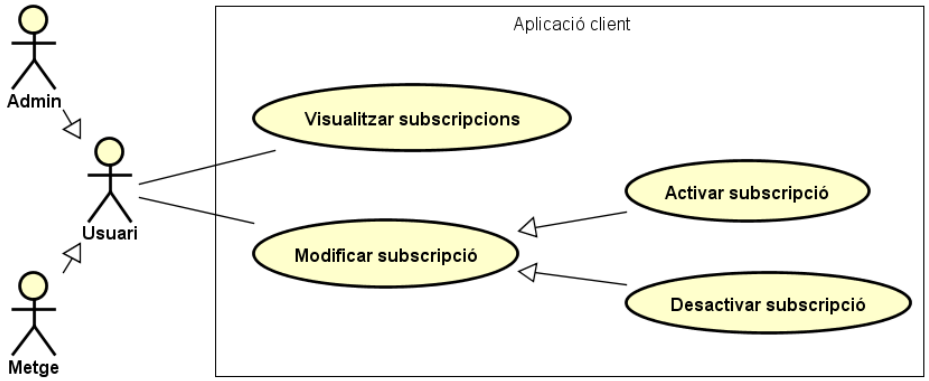

Figure 4.20: "Subscriptions" scenario UML diagram

View active subscriptions (CU-APP-SUB-01)

CU-APP-SUB-01	*View active subscriptions*
Actors	Physician - Admin
Description	The user wants to view her active subscriptions. Accesses to the subscriptions screen from the main screen menu. If the user is not an administrator, the application will use the session user name to get the subscriptions; if she is, the application will request a user name to get the subscriptions list. The subscriptions will be filtered by meta-alarm identifier.
Pre-conditions	
Pre-condition 1	The application has started a user session
Main scenario	*Subscriptions listed successfully*
Step 1	The user name is sent to the **server** (*CU-SER-SUB-01*)
Step 2	The server returns the requested subscription list
Step 3 (Final)	The **application** show the list to the user
Variation A	*No subscriptions found*
Step 2	The server returns an empty list
Step 3 (Final)	The application tells the user that no results have been found
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3 (Final)	The application informs the user about the error

Table 4.22: CU-APP-SUB-01 description

The sequence diagram related to the use-case is the following:

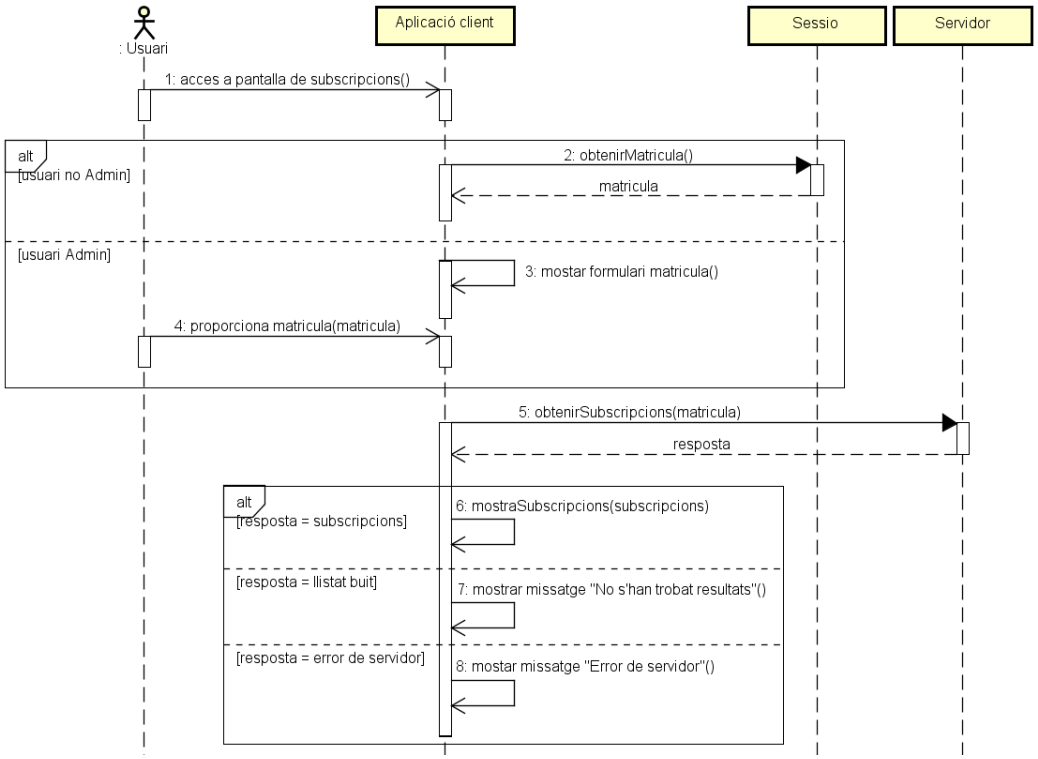

Figure 4.21: SEQ-APP-SUB-01 diagram

Update subscriptions (CU-APP-SUB-02)

CU-APP-SUB-02	*Update subscriptions*
Actors	Physician - Admin
Description	The user wants to activate/deactivate her subscriptions. In the subscription list screen, presses the "Edit" button and selects the subscriptions that will be modified. The subscriptions are filtered by meta-alarm identifier. Once the subscriptions have been modified, the user presses the "Save" button and the application goes back to the subscription list screen.
Pre-conditions	
Pre-condition 1	The application has started a user session
Pre-condition 2	The user subscriptions have been listed
Main scenario	*Subscriptions updated successfully*
Step 1	The user name and the modifications are sent to the **server** (*CU-SER-SUB-02*)
Step 2	The server returns a successful response

Step 3 (Final)	The application informs the user that the update has been successful
Variation A	*Update failure*
Step 2	The server returns a response saying that subscriptions have not been updated
Step 3 (Final)	The application informs the user that the update has not been successful
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3 (Final)	The application informs the user about the error

Table 4.23: CU-APP-SUB-02 description

The sequence diagram related to the use-case is the following:

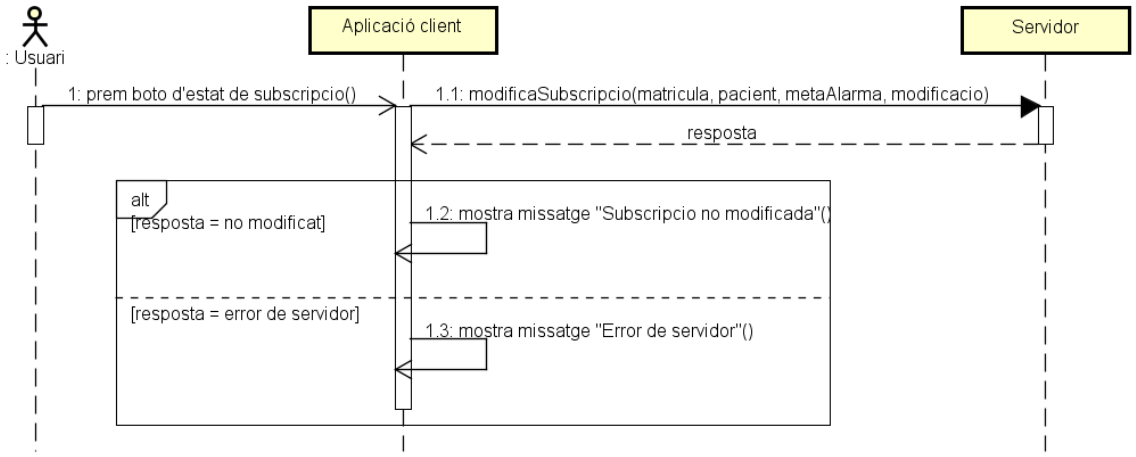

Figure 4.22: SEQ-APP-SUB-02 diagram

Notifications (NOT)

Scenario related to the notification management done by the user into the application. The UML diagram from the scenario is the following:

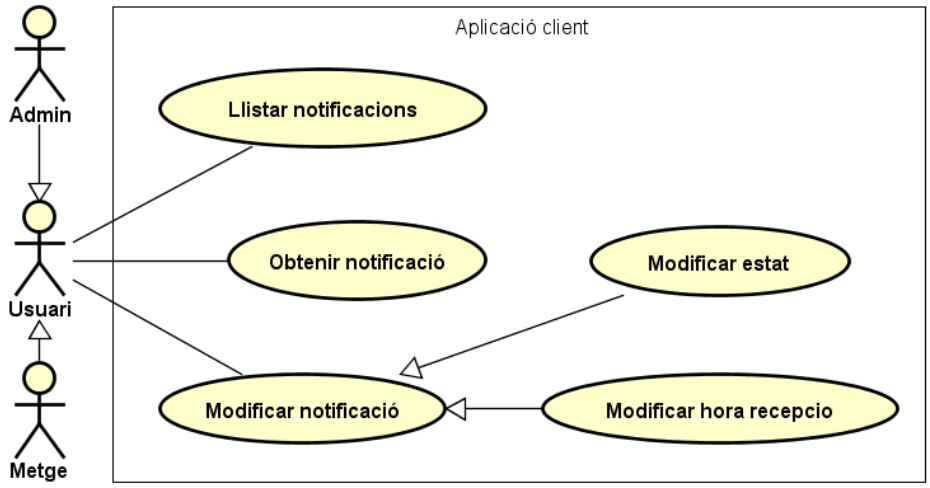

Figure 4.23: "Notifications" scenario UML diagram

View notifications (CU-APP-NOT-01)

CU-APP-NOT-01	*View notifications*
Actors	Physician - Admin
Description	The user wants to view her notifications. Accesses to the notifications screen from the main screen menu. If the user is not an administrator, the application will use the session user name to get the notifications; if she is, the application will request a user name to get the notifications list. The notifications will be filtered by its status.
Pre-conditions	
Pre-condition 1	The application has started a user session
Main scenario	*Notifications listed successfully*
Step 1	The user name is sent to the **server** (*CU-SER-NOT-01*)
Step 2	The server returns the notifications list
Step 3 (Final)	The **application** shows the list to the user
Variation A	*No notifications found*
Step 2	The server returns an empty list
Step 3 (Final)	The application tells the user that no results have been found
Variation B	*Server error*

Step 2	The server returns an HTTP server error response
Step 3 (Final)	The application informs the user about the error

Table 4.24: CU-APP-NOT-01 description

The sequence diagram related to the use-case is the following:

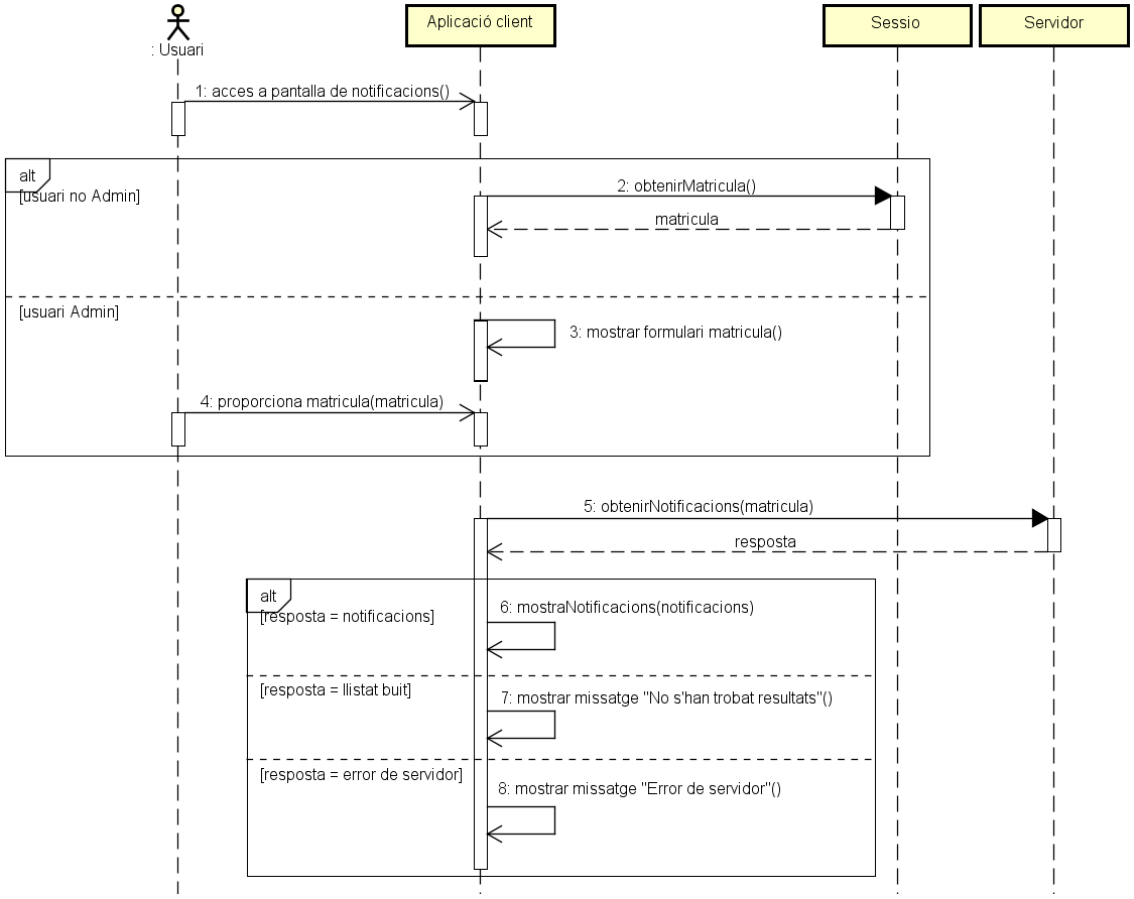

Figure 4.24: SEQ-APP-NOT-01 diagram

Notification details (CU-APP-NOT-02)

CU-APP-NOT-02	*Notification details*
Actors	Physician - Admin
Description	The user wants to see the details of a notification by selecting it from the list
Pre-conditions	
Pre-condition 1	The application has started a user session
Pre-condition 2	The notifications from a user have been listed

Main scenario	*Details shown successfully*
Step 1	The user name and notification identifier are sent to the **server** (*CU-SER-NOT-03*)
Step 2	The server returns the requested notification details
Step 3 (Final)	The application shows the **notification** details to the user
Variation A	*No content found*
Step 2	The server doesn't return any notification data
Step 3 (Final)	The application informs the user the notification data has not been found
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3 (Final)	The application informs the user about the error

Table 4.25: CU-APP-NOT-02 description

The sequence diagram related to the use-case is the following:

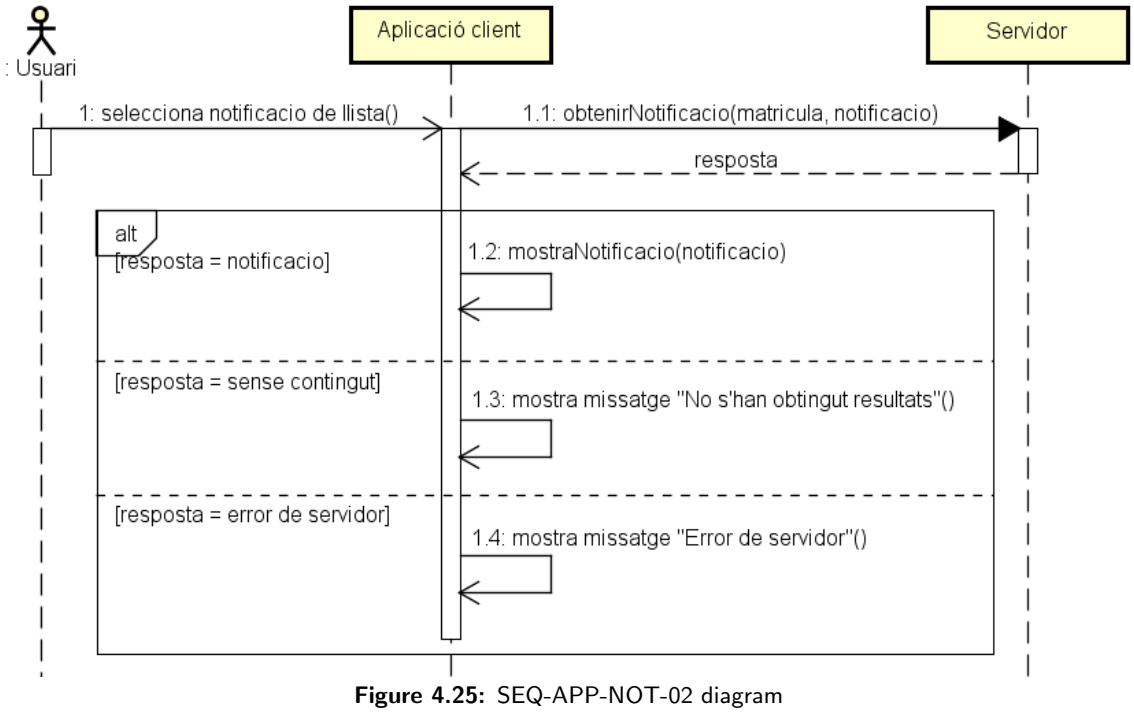

Figure 4.25: SEQ-APP-NOT-02 diagram

Update notification (CU-APP-NOT-03)

CU-APP-NOT-03	*Update notification*
Actors	Physician - Admin
Description	The user wants to update the status of a notification by clicking the edit button in the notification details. A dialog appears to allow the user to change the status and save the modification.
Pre-conditions	
Pre-condition 1	The application has started a user session
Pre-condition 2	The details of a notification have been obtained
Main scenario	*Notification updated successfully*
Step 1	The user name, the notification identifier and the changes are sent to the **server** (*CU-SER-NOT-02*)
Step 2	The server responses successfully
Step 3	The application informs the user he notification has been modified
Step 4 (Final)	The application shows the modified notification to the user
Variation A	*Update failure*
Step 2	The server returns an update error
Step 3	The application informs the user the notification has not been updated
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3	The application informs the user about the error

Table 4.26: CU-APP-NOT-03 description

The sequence diagram related to the use-case is the following:

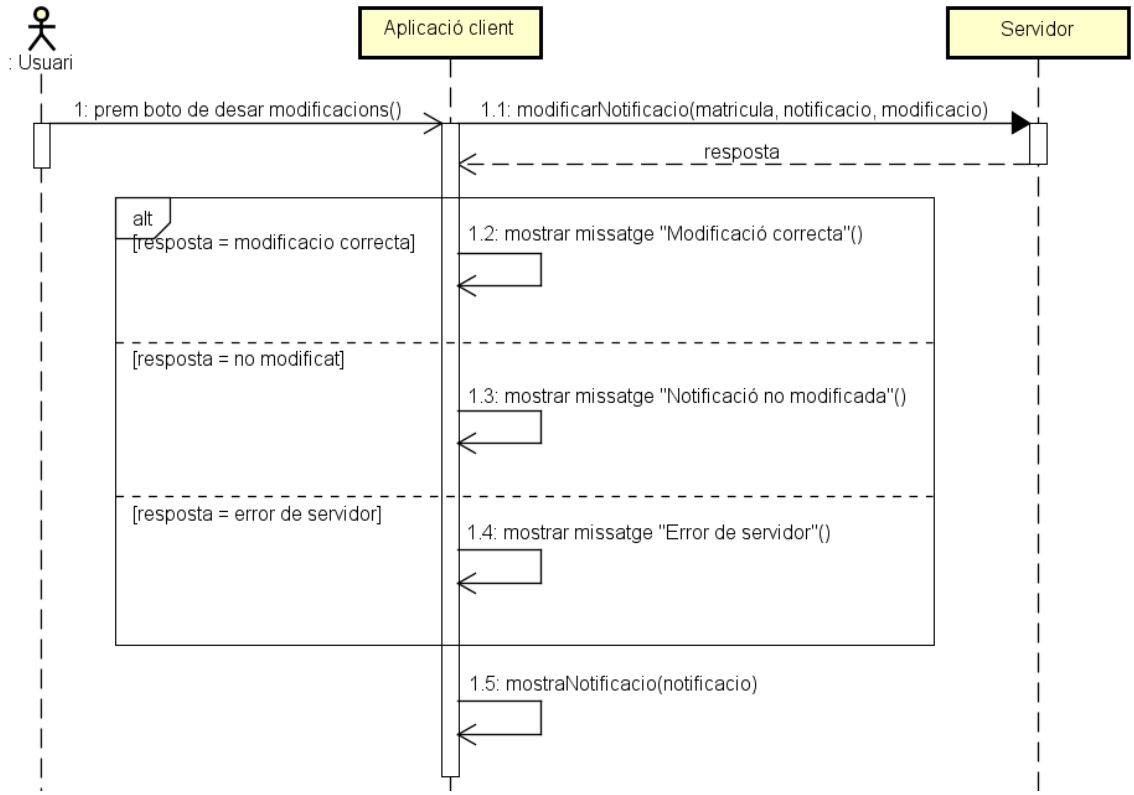

Figure 4.26: SEQ-APP-NOT-03 diagram

Push notifications (NPU)

Scenario related to the management of the notification arrival into the device. This scenario is only available for the Android application. The UML diagram from the scenario is the following:

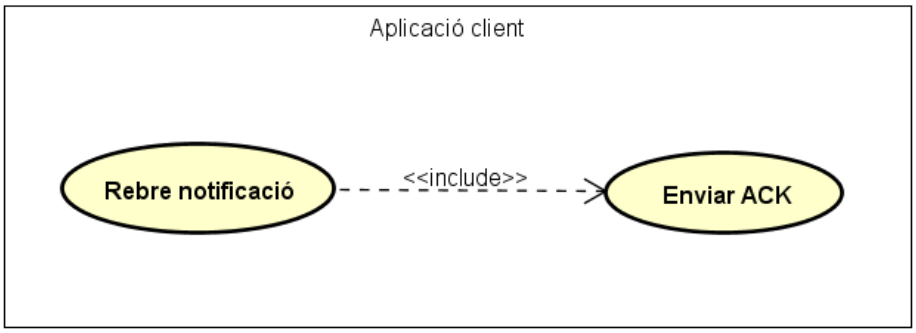

Figure 4.27: "Push notifications" scenario UML diagram

Receive notification (CU-APP-NPU-01)

CU-APP-NPU-01	*Receive notification*
Actors	
Description	The application receives a push notification sent from SCAS through Firebase Cloud Messaging platform
Pre-conditions	
Pre-condition 1	The application has started a user session
Main scenario	*Notification received correctly*
Step 1	FCM sends a notification to the device
Step 2	The application decrypts the notification
Step 3	The application shows the push notification to the user
Step 4 (Final)	The application sends an ACK to the server (*CU-APP-NPU-02*)

Table 4.27: CU-APP-NPU-01 description

The sequence diagram related to the use-case is the following:

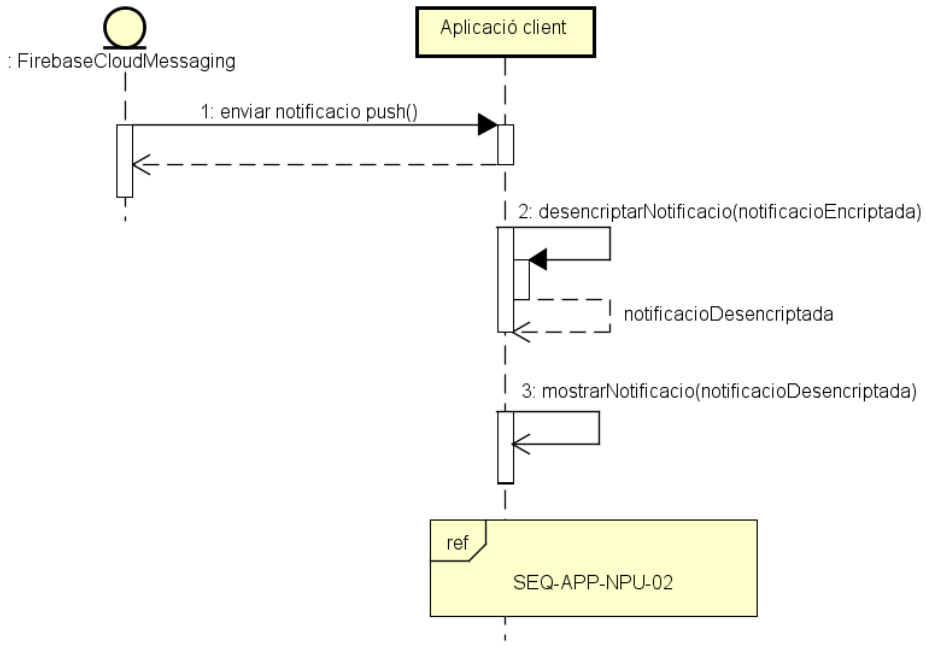

Figure 4.28: SEQ-APP-NPU-01 diagram

Send ACK (CU-APP-NPU-02)

CU-APP-NPU-02	*Send ACK*
Actors	
Description	The application confirms to SCAS the notification has been received into the device
Pre-conditions	
Pre-condition 1	The application has started a user session
Main scenario	*ACK sent correctly*
Step 1	The application gets the user name from the session
Step 2 (Final)	The application sends the user name and the notification identifier that wants to confirm to the server

Table 4.28: CU-APP-NPU-02 description

The sequence diagram related to the use-case is the following:

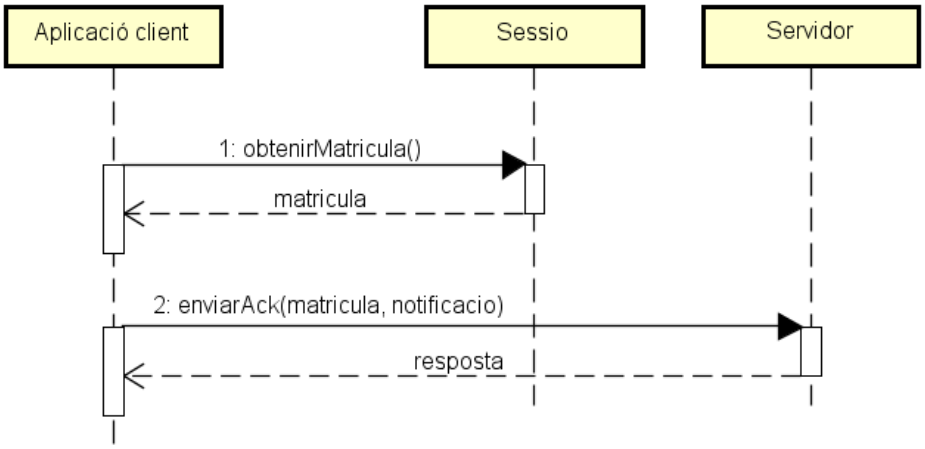

Figure 4.29: SEQ-APP-NPU-02 diagram

Users (USR)

Scenario related to user management done by the administrator into the application. This scenario is only available on the web application. The UML diagram from the scenario is the following:

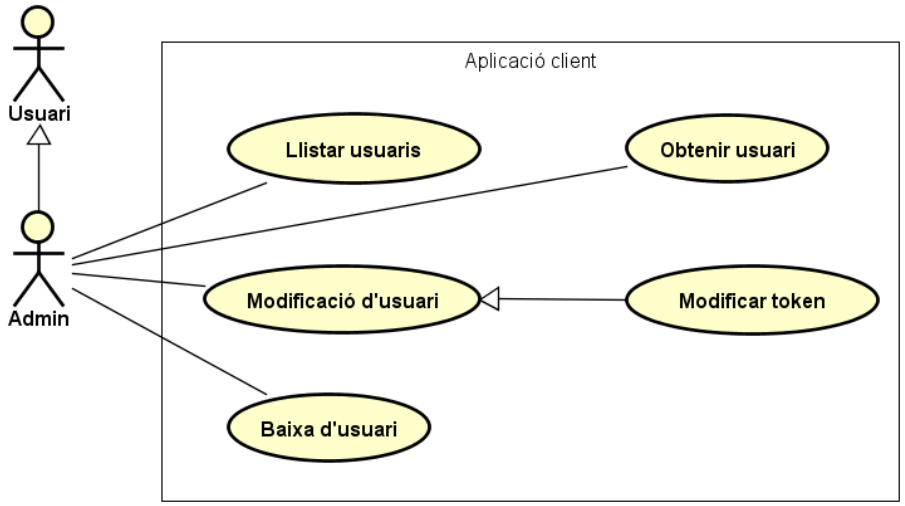

Figure 4.30: "Users" scenario UML diagram

List users (CU-APP-USR-01)

CU-APP-USR-01	*List users*
Actors	Admin
Description	The administrator wants to view the list of non-admin users from the system. Accesses to the user screen from the main screen menu
Pre-conditions	
Pre-condition 1	The application has started a user session with administrator role
Main scenario	*Users listed successfully*
Step 1	The application requests the user list to the **server** (*CU-SER-USR-01*)
Step 2	The server returns the users list
Step 3 (Final)	The application shows the list to the user
Variation A	*No content*
Step 2	The server returns an empty list
Step 3 (Final)	The server informs the user that no users have been found
Variation B	*Server error*
Step 2	The server returns an HTTP server error response

Step 3 (Final)	The application informs the user about the error

Table 4.29: CU-APP-USR-01 description

The sequence diagram related to the use-case is the following:

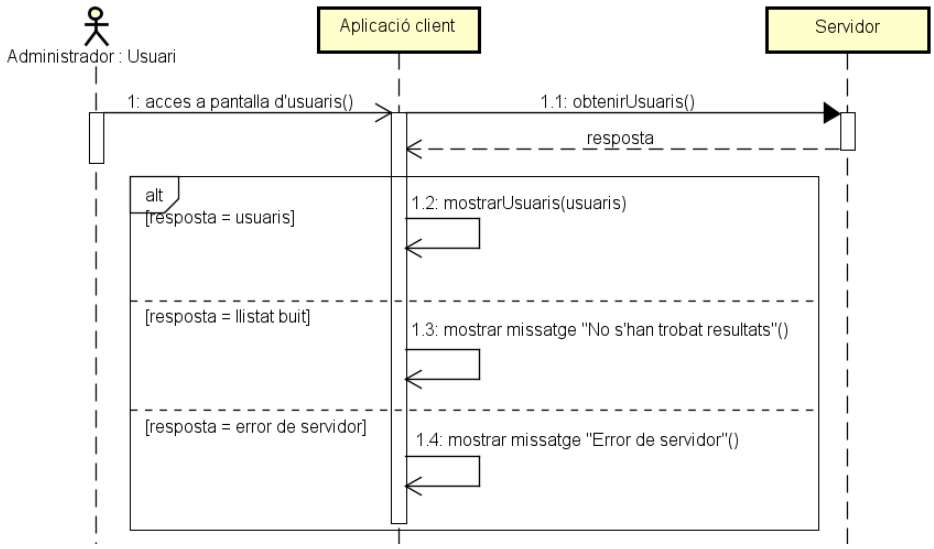

Figure 4.31: SEQ-APP-USR-01 diagram

User details (CU-APP-USR-02)

CU-APP-USR-02	*User details*
Actors	Admin
Description	The administrator wants to see the details from a user by selecting it from the list
Pre-conditions	
Pre-condition 1	The application has started a user session with administrator role
Pre-condition 2	The users have been listed
Main scenario	*Details obtained successfully*
Step 1	The user name is sent to the **server** (*CU-SER-USR-05*)
Step 2	The server returns the data related to the user
Step 3 (Final)	The application shows the details to the user
Variation A	*No contents*
Step 2	The server doesn't return user data
Step 3	The application informs the user data has not been found
Variation B	*Server error*

Step 2	The server returns an HTTP server error response
Step 3	The application informs the user about the error

Table 4.30: CU-APP-USR-02 description

The sequence diagram related to the use-case is the following:

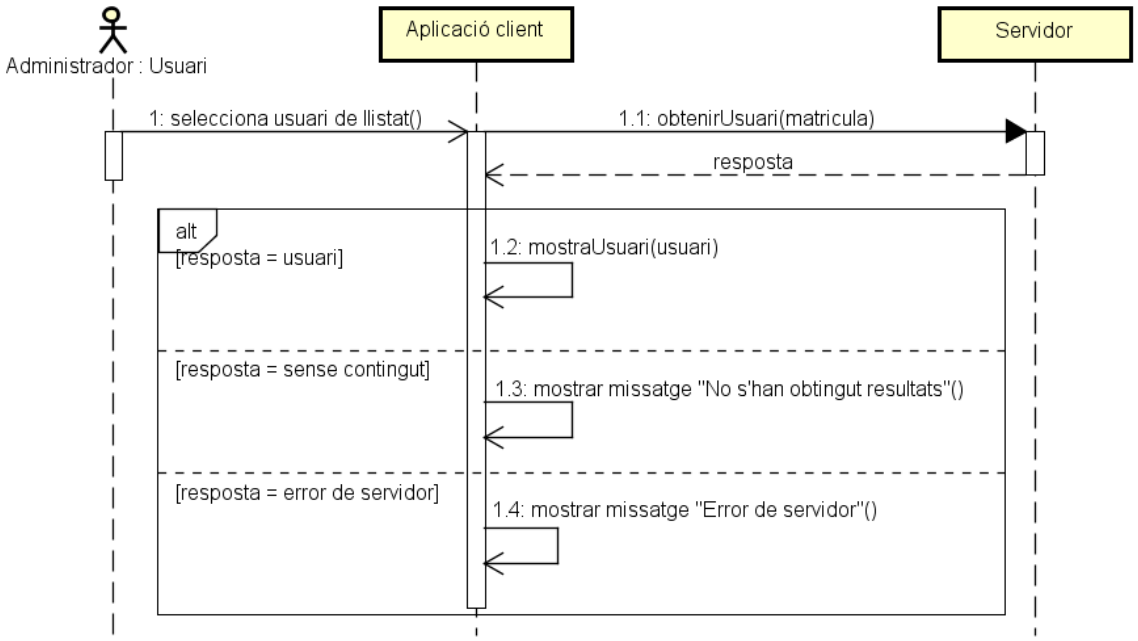

Figure 4.32: SEQ-APP-USR-02 diagram

Update user (CU-APP-USR-03)

CU-APP-USR-03	*Update user*
Actors	Admin
Description	The administrator wants to modify data from a user. Accesses the user edit screen, makes changes and saves them
Pre-conditions	
Pre-condition 1	The application has started a user session with administrator role
Pre-condition 2	The user details have been obtained
Main scenario	*User updated successfully*
Step 1	The user name and the changes are sent to the **server** (*CU-SER-USR-04*)
Step 2	The server responses successfully
Step 3	The application informs the user that data has been updated

Step 4 (Final)	The application shows the user details
Variation A	*User update failure*
Step 2	The server responses unsuccessfully response saying
Step 3	The application informs the update has not been correct
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3	The application informs the user about the error

Table 4.31: CU-APP-USR-03 description

The sequence diagram related to the use-case is the following:

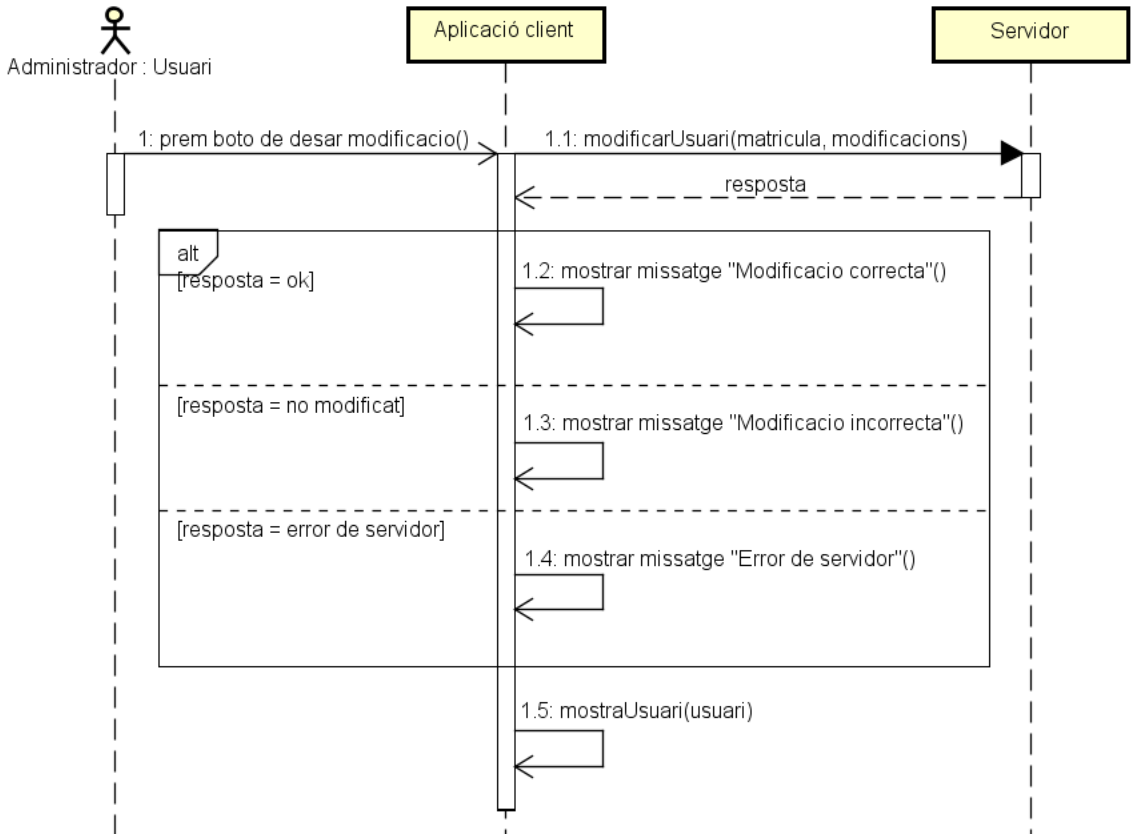

Figure 4.33: SEQ-APP-USR-03 diagram

Delete user (CU-APP-USR-04)

CU-APP-USR-04	*Delete user*
Actors	Admin
Description	The administrator wants to delete a user. Presses the delete user button and confirms the removal

Pre-conditions	
Pre-condition 1	The application has started a user session with administrator role
Pre-condition 2	The users have been listed
Main scenario	*User deleted successfully*
Step 1	The user name is sent to the **server** (*CU-SER-USR-03*)
Step 2	The server responses successfully
Step 3	The application informs that the user has been removed
Step 4 (Final)	The application shows the users list
Variation A	*User not removed*
Step 2	The server responses unsuccessfully
Step 3	The application informs the user that the removal has not been done
Variation B	*Server error*
Step 2	The server returns an HTTP server error response
Step 3	The application informs the user about the error

Table 4.32: CU-APP-USR-04 description

The sequence diagram related to the use-case is the following:

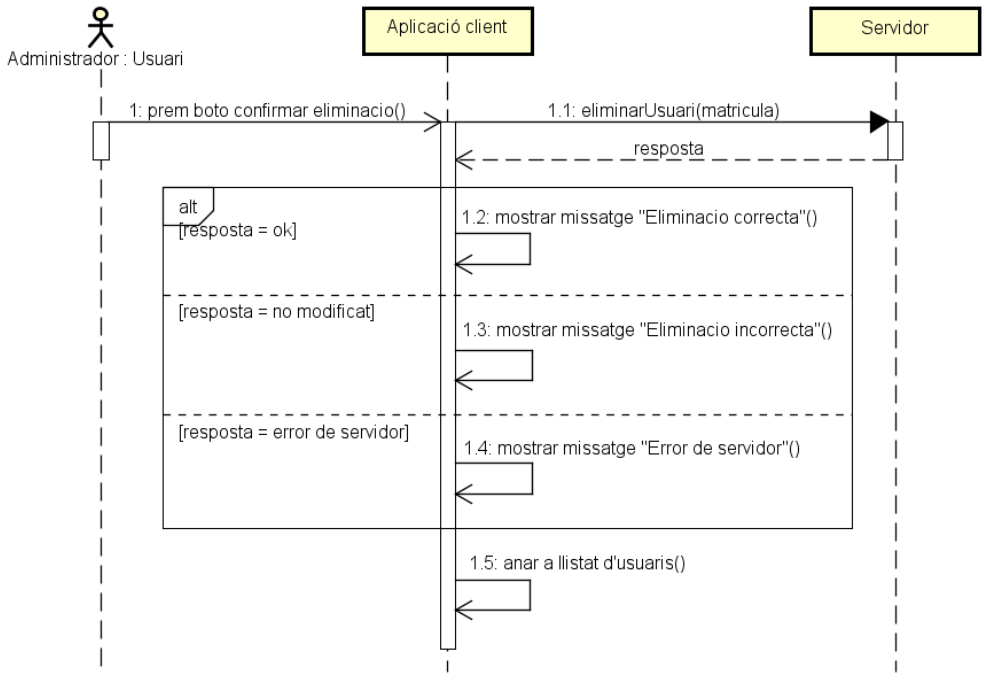

Figure 4.34: SEQ-APP-USR-04 diagram

4.6 Architecture

This section shows an overview of the HIS architecture where the project is going to be implemented and the SCAS architecture. The combination of both architectures is the following:

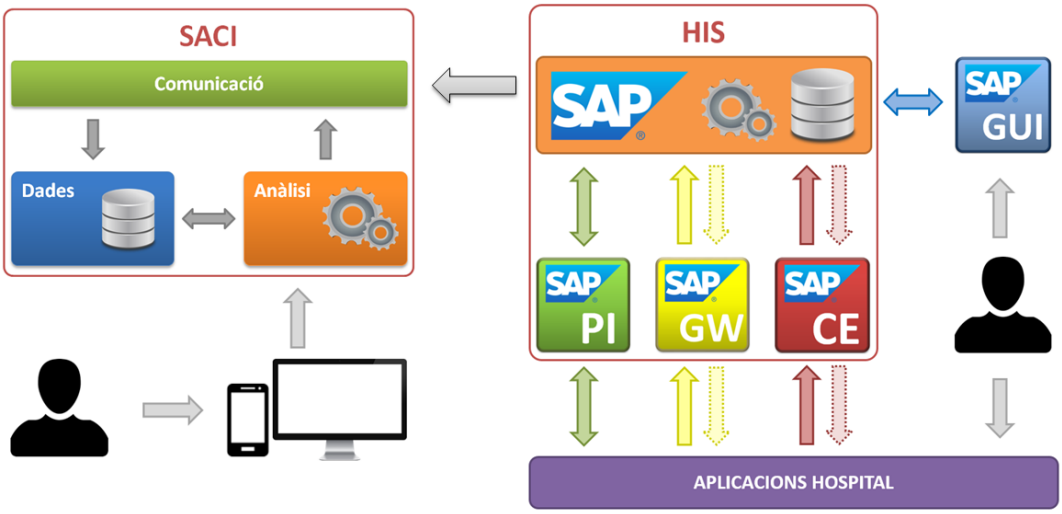

Figure 4.35: Integration architecture

4.6.1 HIS architecture

The HIS architecture is based on SAP, an administrative software composed by the elements shown in the following figure.

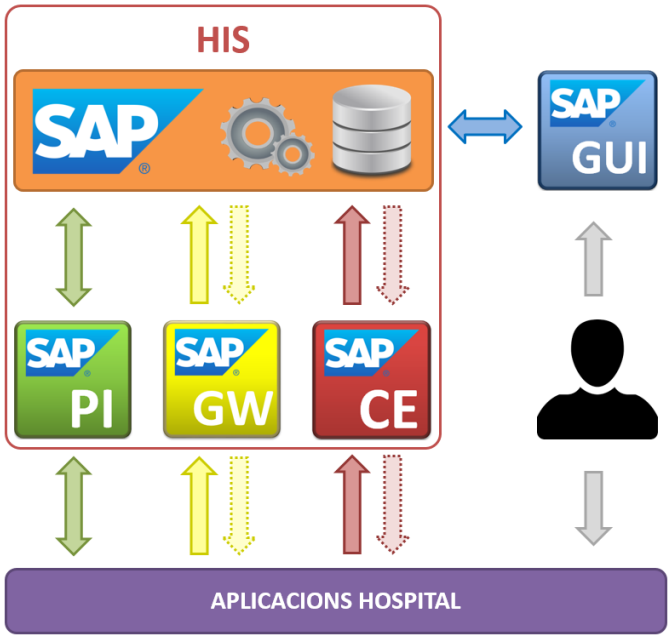

Figure 4.36: HIS architecture

- **SAP:** it's the HIS base system. Its DB, based on Oracle, stores all the clinical information from the hospital.
- **SAP GUI:** the Graphical User Interface (GUI) allows the user to interact directly with SAP base system.
- **Hospital applications:** consist on the applications developed by the hospital IT department for interacting with SAP indirectly.
- **SAP PI:** Process Integrator (PI) communicates asynchronously SAP with hospital applications. The communication can be started from SAP to applications and vice versa.
- **SAP GW:** Gateway (GW) communicates synchronously SAP with hospital applications. The communication is only started from applications to SAP.
- **SAP CE:** Composition Environment (CE) has the same functionality as GW, but the difference is that GW is used on applications developed on SAPUI5 (a Javascript library for developing web applications) and CE is used on applications developed on Flash.

Some examples on how the different SAP elements work together:

- **GUI - SAP**: The physician searches on **GUI** for patient clinical history and makes changes that will be updated directly on **SAP**.
- **Application - GW - SAP:** A nurse, using a tablet, accesses to an application to evaluate the patient status (temperature, blood pressure, oxygen saturation...).
  - The **application** makes a query to **GW** for getting or modifying data
  - The **GW** sends the query to **SAP**
  - **SAP** responses to **GW**
  - **GW** sends the response back to the **application**
  - The **application** shows the response to the nurse
- **Application - PI - SAP:** A patient gets a ticket to know her turn.
  - The **application** sends the patient's turn to **PI**
  - **PI** send the turn to **SAP** and SAP stores it
  - The physician, using the **GUI**, selects the patient's turn to whom wants to visit
  - **SAP** sends to **PI** the physician decision
  - **PI** sends the decision back to the **application**
  - The chosen patient gets visited by the physician
- **SAP - PI - Application:** A physician requests some laboratory tests for a patient.
  - The physician, using the **GUI**, makes the request
  - **SAP** stores the request and sends it to **PI**

- **PI** sends the request to the laboratory **application**
- The laboratory technician gets the request and can start to make the tests

4.6.2 SCAS architecture

The SCAS architecture is based on REST Web Services and is composed by the elements shown in the following figure.

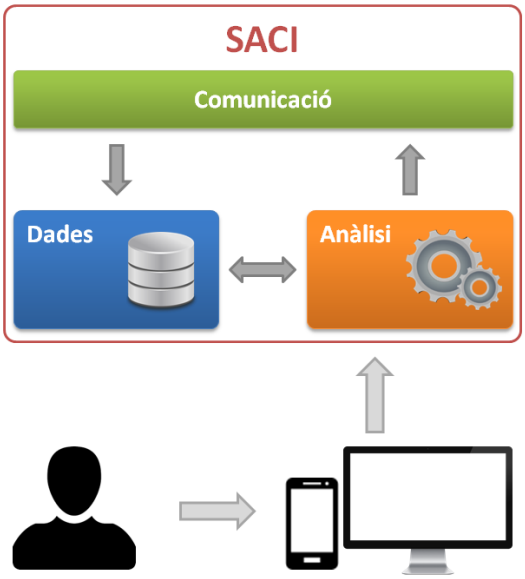

Figure 4.37: SCAS architecture

- **Communication:** communicates with HIS for getting clinical data and sends the notifications in case of patient risk.
- **Analysis:** checks HIS clinical data in order to find whether a patient is in risk or not.
- **Data:** stores users, alarm definitions and the notifications generated by the service after the HIS data has been checked.

Following example shows how the different elements work together:

- **Communication** gets HIS clinical data and sends it to **Analysis**.
- **Analysis** processes the received data (checking the definitions from **Data**) and, if an alarm is detected, does the following:
  - Generates a notification and tells **Data** to store it.
  - Tells **Communication** to send the notification to the users.

5 System implementation

5.1 Server implementation

The server side of the SCAS has been implemented on a Linux server using Java, Tomcat and PostgreSQL. The specifications of the server are listed below.

- Operative System: SUSE Linux Enterprise Server 12 (x86_64)
- Hard-disk drive size: 40 GB
- RAM size: 4 GB
- Software specifications
  - Java: OpenJDK version 1.8.0_121
  - Tomcat: Apache Tomcat/8.0.36
  - PostgreSQL: version 9.4.9

5.1.1 Java code

The full structure of the Java code that has been created for building the server functionality is shown in figures 5.1 and 5.2.

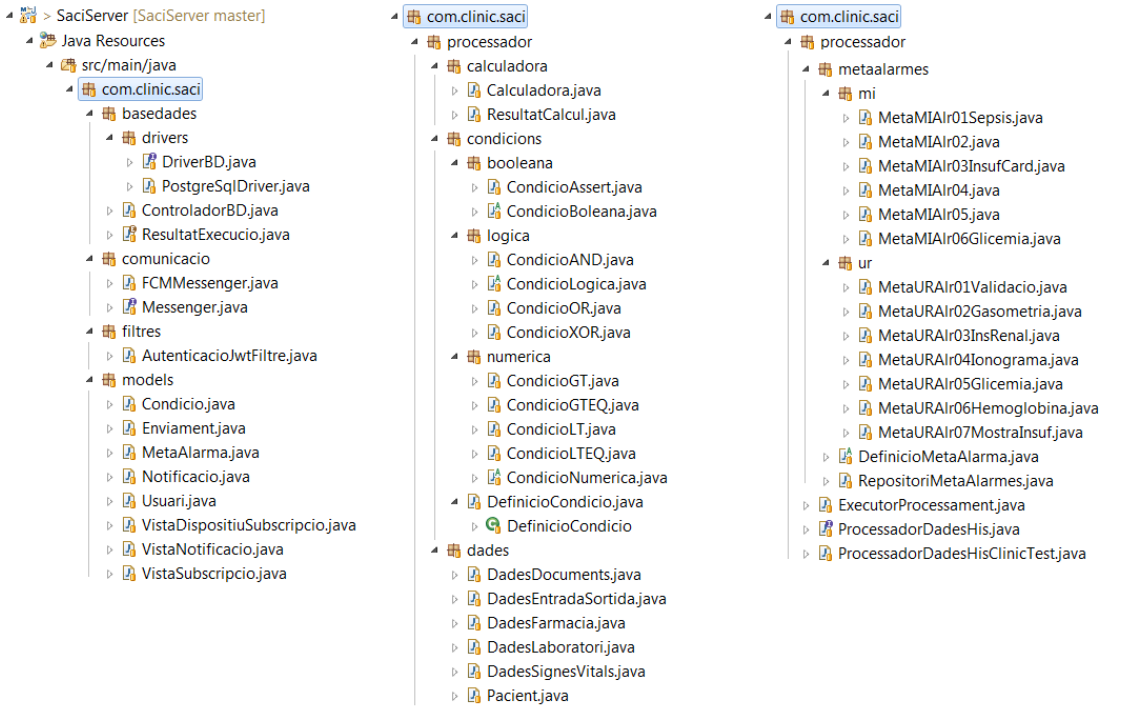

Figure 5.1: Java server side code (1)

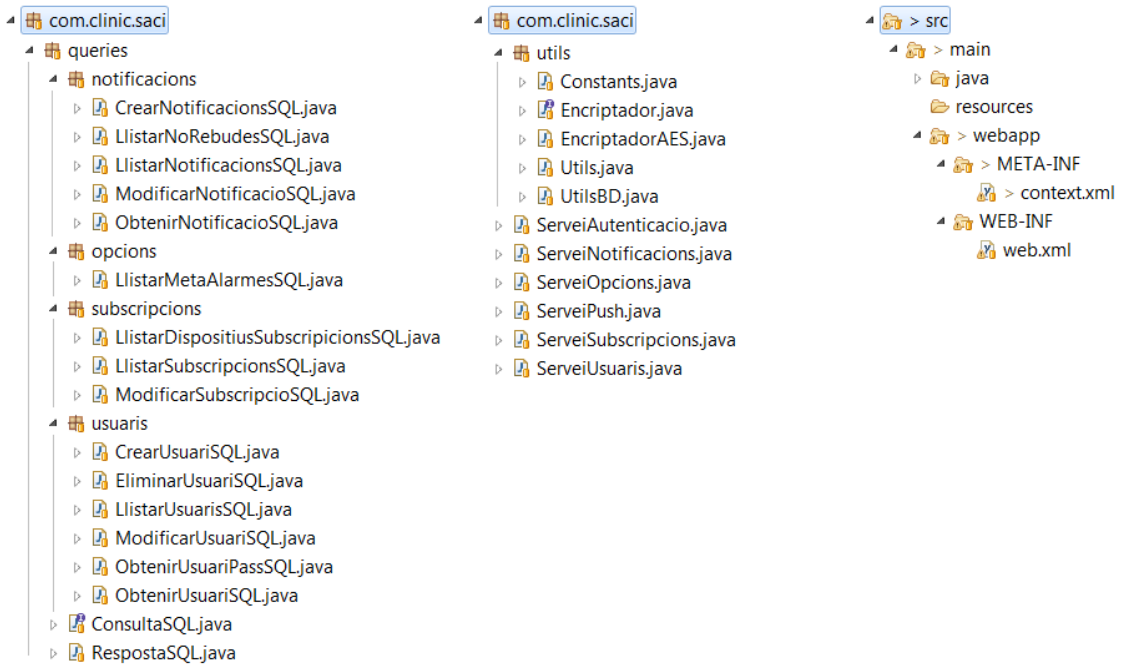

Figure 5.2: Java server side code (2)

Even though it is a big structure (almost 90 classes), the amount of lines is around 10K, so the mean of lines for each class is 100, which is not too much. The reason why there are so many classes is because of the **SOLID principles**. The fact of trying to give single responsibilities to classes, using interfaces and abstractions as dependencies and making the code as closed as possible to changes has come into the creation of the amount of classes mentioned before. Some of the most relevant classes are explained in the next lines.

Servei[service] classes

These classes expose the methods that are used for interact with the system data using Web Services. In order to expose those methods, the Jersey [20] library, which provides APIs to build RESTful Web Services using Java, has been used. To expose a WS there are needed (1) an XML servlet definition (located in WEB-INF/web.xml), (2) the Java class to expose and (3) the tags defining the path to the method and the HTTP method that will be used.

In **web.xml** it has to be included which package contains the classes that are going to be exposed. Then, the Java class uses tags like @Path (to define which URL will map to the Java method), @POST/@GET/@PUT (to define which HTTP method needs to be used) and @Consumes/@Produces (to define which kind of data the HTTP method will give and/or receive) that will be interpreted by Tomcat when the service is deployed on it.

```
<!-- Servlet -->
<servlet>
    <servlet-name>rest-servlet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>com.clinic.saci</param-value>
    </init-param>
```

```
@Path("/autenticacio")
public class ServeiAutenticacio {

    @SuppressWarnings("unchecked")
    @POST
    @Path("/login")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response obtenirUsuari(@Context UriInfo info, Usuari usuari){
```

Figure 5.3: web.xml and ServeiAutenticacio.java snippet

"utils" package

This package contains functionalities that can be used by all the other classes. The most relevant class from the package is **EncriptadorAES.java** which is used for encrypt the data from the notifications that are going to be sent through Firebase Cloud Messaging. The reason of the encryption of this data is that, as they are private and sensitive, it's a risk to send them in clear outside the hospital. If anyone who wants this kind of data for malicious purposes is able to intercept and read the message, the patient security could be compromised.

The other problem is that, even if the cloud service used is supposed to be secure, no one can guarantee that the data being sent is not being stored in an external server for the own cloud service company benefits. That said, the purpose of this class is to bring security to the SCAS using an **Advanced Encryption Standard (AES)** algorithm which was approved by the National Security Agency (NSA) for securing top secret information.

"queries" package

This package contains all the SQL queries that are requested to the PostgrSQL DB. All of them are defined by the **ConsultaSQL.java** interface, which is very simple, as shown in figure 5.4. The classes implementing this interface have to (1) define the query when the object is created, (2) override the "executa" method to connect, send the request and disconnect from DB and (3) override the "obtenirResultat" method to return the results from the query.

```
public interface ConsultaSQL {
    public void executa();
    public RespostaSQL obtenirResultat();
}
```

```
public class RespostaSQL<T> {

    private int codi;
    private String missatge;
    private List<T> contingut;
}
```

Figure 5.4: ConsultaSQL and RespostaSQL

The generic class **RespostaSQL.java** is used to get the results from the queries. It's composed by the HTTP code that will be returned once the query has been executed, a message containing additional info about the query execution and the list of objects that have been obtained from the query.

"models" package

This package contains the Java representations of the data that has to be obtained from the DB. None of the classes are so relevant to be explained as they are just Plain Old Java Object (POJO).

"filtres" package

This package contains one class (**AutenticacioJwtFiltre.java**) which functionality is checking if the header of the HTTP request contains a valid authentication field. This field is provided by **ServeiAutenticacio.java** when the user has authenticated successfully and contains a Json Web Token (JWT).

JWT [21] is a method for securely exchange information between two parties. In this case, the server generates the token and sends it to the client. Then, the client needs to include this token on each request to the server in order to verify that the client is allowed to obtain data. The token is generated as a Hash-based Message Authentication Code (HMAC) using a Secure Hash Algorithm (SHA) of 512 bits.

This is another security mechanism that has been added to SCAS in order to create a system which is as secure as possible.

"comunicacio" package

This package contains the necessary classes for sending notifications through FCM. The **Messenger.java** interface defines the methods that will be implemented by **FCMMessenger.java**.

```
public interface Messenger {
    public void crearContingut(Notificacio notificacio, String destinatari);
    public void enviarContingut();
}
```

Figure 5.5: Messenger interface

As shown in figure 5.5, the FCMMessenger class will receive the notification that must be sent and the destination of it. The destination refers to the token of the device that will receive the push notification. This token is obtained when the user is successfully authenticated from the Android application.

"basedades" package

This package contains the drivers and controllers used for interacting with the DB. The **DriverBD.java** interface defines the main methods that will be used for connecting and disconnecting to the DB and search or modify data into it. The **ControladorBD.java** can be considered as some kind of bridge between the SQL queries and the driver to make easier the exception handling.

```java
public class ControladorBD {

    /* Driver de la BD */
    private final DriverBD driverBD;

    /* Constructor */
    public ControladorBD(DriverBD driverBD) {
        this.driverBD = driverBD;
    }

    /* Mètodes comuns */

    public void establirConnexio() throws ExcepcioConnexioBD{
        try {
            driverBD.connectar();
        } catch (SQLException e) {
            throw new ExcepcioConnexioBD(e);
        }
    }

    public void finalitzarConnexio() throws ExcepcioDesconnexioBD{
        try {
            driverBD.desconnectar();
        } catch (SQLException e) {
            throw new ExcepcioDesconnexioBD(e);
        }
    }

    public ResultSet queryConsulta(String sqlQuery) throws ExcepcioConsultaBD{
        try {
            return driverBD.executarQueryConsultora(sqlQuery);
        } catch (SQLException e) {
            throw new ExcepcioConsultaBD(e);
        }
    }
}
```

```java
public interface DriverBD {

    public void modificarParametresConnexio(String url, String usuari, String contrasenya);
    public void connectar() throws SQLException;
    public ResultSet executarQueryConsultora(String query) throws SQLException;
    public void executarQueryModificadora(String query) throws SQLException;
    public void desconnectar() throws SQLException;

}
```

Figure 5.6: Controller and driver implementations

The main reason of this is that the driver only returns one kind of exception and the final queries' code would be full of try-catch statements in order to manage them when sending requests to the DB, which makes the code more difficult to read and understand. Using the controller, as each method is managing the driver exceptions separately, makes easier to manage which exception is raised in every part of the queries' code and everything can be handled in a single try-catch clause.

```java
@Override
public void executa() {

    try {

        /* Connectar amb la BD */
        controlador.establirConnexio();
        /* Obtenir el resultat */
        ResultSet rs = controlador.queryConsulta(sqlQuery);
        /* Desconnectar de la BD */
        controlador.finalitzarConnexio();

    } catch (ExcepcioConnexioBD e) {

    } catch (ExcepcioDesconnexioBD e) {

    } catch (ExcepcioConsultaBD e) {

    }
}
```

Figure 5.7: Example of the DB exceptions handling using the controller

"processador" package

This package contains the necessary elements for processing HIS data. The packages contained in it are described below.

The **"dades" package** contains POJOs that will act as the models of the data that is obtained from the HIS.

The **"condicions" package** contains the definitions of the different conditions that will be checked in the meta-alarms. There are three types of conditions, as shown in figure 5.8. Each condition is defined by the value that will have if they have been accomplished. Numerical conditions check if a number is greater or lower than another number and boolean conditions check if a value is true or false, so they are very simple. On the other side, logical conditions can compare two conditions, which can be numerical, boolean or even logical conditions, so they are a little more complex than the other ones.

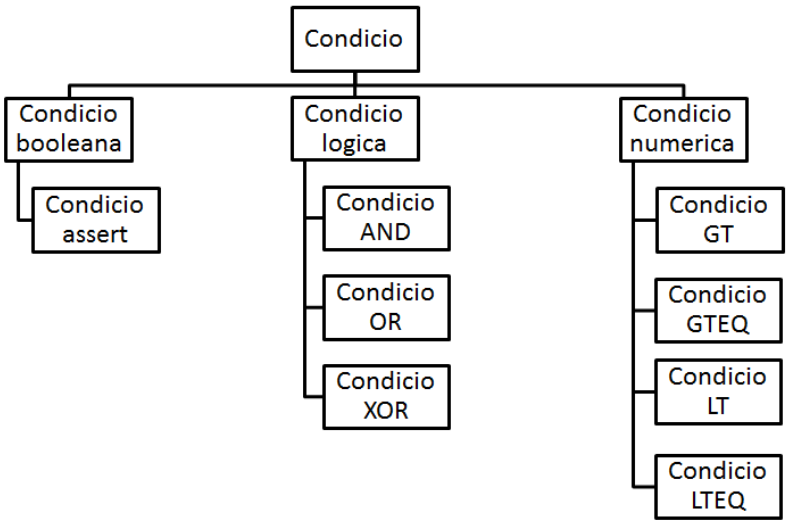

Figure 5.8: Types of conditions

The **"meta-alarmes" package** contains the definitions on how the meta-alarms are going to be calculated. Each meta-alarm is composed by (1) a list of conditions, from the "condicions" package, that will be checked using the patient data and (2) the value that the accomplished conditions need to reach in order to consider that the patient presents some risk.

Assuming the Sepsis definition on table 3.5, the meta-alarm expected value is 3 and contains 14 conditions, each of them with a value of 1 or 2 depending on the condition. After checking the patient data, if the accomplished conditions added value is equal or greater than the meta-alarm expected value, an alarm will be generated.

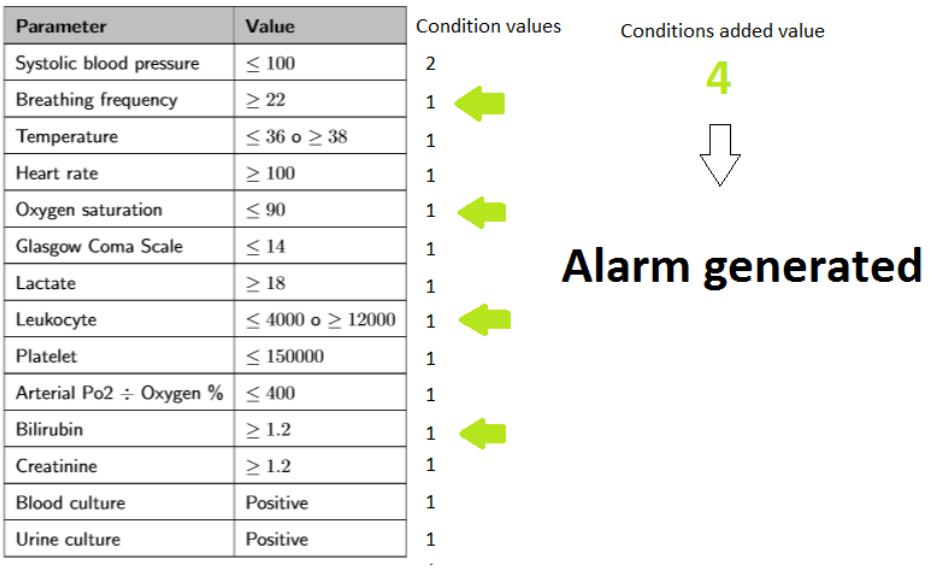

Figure 5.9: Graphical example of the meta-alarms computation

5.1.2 Tomcat configuration

Enabling HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is the secure version of HTTP. It uses a Transport Layer Security (TLS) certificate between client and server in order to encrypt the data when are sent over the internet and decrypt them when arrive to the destination. In this way, attackers are unable to read the data if they intercept the connection.

In order to add this security protocol to the SCAS, few steps have been done:

- Generate and sign a TLS certificate and store it in a keystore on the server
- Modify **server.xml** file, located in the Tomcat installation directory, to enable HTTPS specifying where the certificate has been placed
- Modify **web.xml** file, located in the SCAS WEB-INF directory, to add a restriction of using HTTPS only.

server.xml

```
<Connector SSLEnabled="true" clientAuth="false"
    keystoreFile="[redacted]" keystorePass="[redacted]"
    maxThreads="150" port="8443" protocol="HTTP/1.1" scheme="https"
    secure="true" sslProtocol="TLS" />
```

web.xml

```
<!-- Security constraint -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HTTPSOnly</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Figure 5.10: server.xml and web.xml modifications

DB connection pool

A connection pool allows to handle multiple connections to a DB at the same time. In cases where only one client needs to connect to a database, the connection can be opened directly and it will be freed once the client has stopped working with it. But in cases where there's more than one user, creating multiple connections can become a resource consumption problem. The connection pool allows to reuse connections when they are free.

The steps for configuring the connection pool are the following:

- Modify **context.xml** file, located in the SCAS META-INF directory, to create a resource with the connection parameters that will define the pool
- Modify **web.xml** file, located in the SCAS WEB-INF directory, to make the resource created in context.xml available to the SCAS components

```
<Resource name="jdbc/SaciDB" auth="Container" type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver" factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
    initialSize="10" maxActive="1000" maxIdle="50"
    minEvictableIdleTimeMillis="60000" minIdle="10" suspectTimeout="60"
    timeBetweenEvictionRunsMillis="30000" url="jdbc:postgresql:[redacted]"
    username="[redacted]" password="[redacted]" />
```

```
<!-- Resource ref -->
<resource-ref>
    <description>Recurs per a la connexió amb la BD</description>
    <res-ref-name>jdbc/SaciDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>res-sharing-scope</res-sharing-scope>
</resource-ref>
```

Figure 5.11: context.xml and web.xml modifications

5.2 Client implementation

The client side of the SCAS has been implemented in AngularJs for the web application and in Android for the mobile application.

5.2.1 AngularJs code

The web application has been implemented in the server mentioned in the previous section to be accessible using a web browser. The code structure is shown in figure 5.12.

The total number of files is around 50 and the amount of lines is around 3000. In this case, as opposed to the server code where each file had more or less the same number of lines, the amount of them depends on the type of file. AngularJs is based on different types of components like modules, factories, services, controllers... In general terms, the ".module" files are the shortest and the ".controller" ones are the largest.

Figure 5.12: AngularJS files distribution

"shared" module

This module contains elements that can be used by the different components of the application. For example, **sessio.factory.js** controls the user session when is using the application and **rest.factory.js** allows performing http requests to retrieve data. This module also contains the **modal** element, which is used for showing message dialogues and it is a little more complex than the other ones as it is a reusable visual element, similar to Android fragments.

"components" folder

This folder contains the different modules corresponding to the different sections of the application. **home** contains the files implementing the main menu from where the user can access to the other sections; **login**, the files implementing the user authentication, and **notificacio**, **subscripcio** and **usuari**, the files implementing the notifications, subscriptions and users sections of the application.

Each component is defined by four files: **view** files contain the graphical representation with which the user will interact directly; **module** files, the declaration of the component into the main application module; **controller** files, the component logic where the user events are handled and the data is processed, and **route** files, the definition of where each part of the application is located.

Figure 5.13: Components relations

"app..js" files and "index.html"

These are the main files for running the application. **index.html** contains all the references to the different libraries and files that should be loaded when the users access to the application. **app..js** files define which modules are used by the application and the functionality of the index.html interface if necessary.

```
<!-- Subscripcions -->
<script src="app/components/subscripcio/subscripcio.module.js"></script>
<script src="app/components/subscripcio/llista/subscripcio-llista.controller.js"></script>
<script src="app/components/subscripcio/edita/subscripcio-edita.controller.js"></script>
<script src="app/components/subscripcio/subscripcio.routes.js"></script>

<!-- App -->
<script src="app/app.module.js"></script>
<script src="app/app.controller.js"></script>
<script src="app/app.routes.js"></script>

<!-- Títol -->
<title>Aplicació web SACI</title>
</head>
```

Figure 5.14: index.html snippet

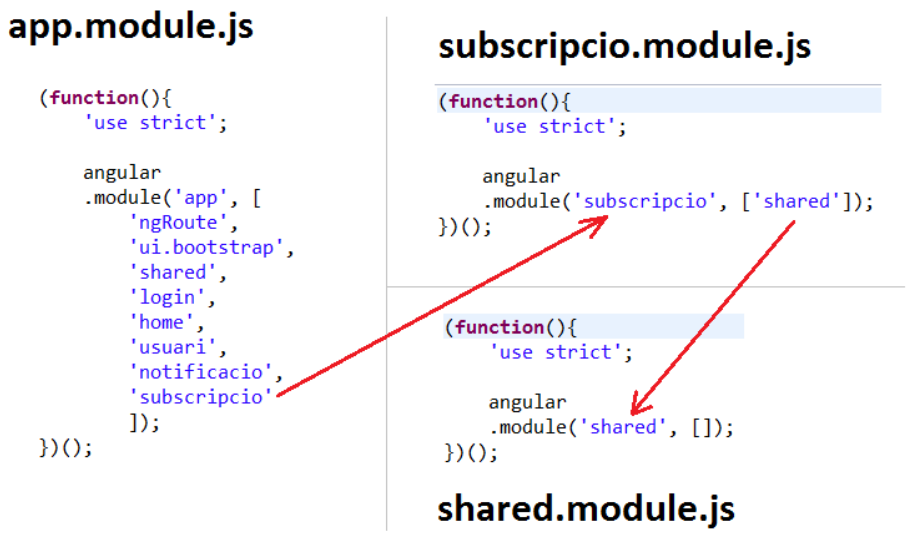

Figure 5.15: Different modules relations

5.2.2 Android code

The mobile application has been implemented for devices using Android version 5.0 or higher. The code structure is shown in figure 5.16.

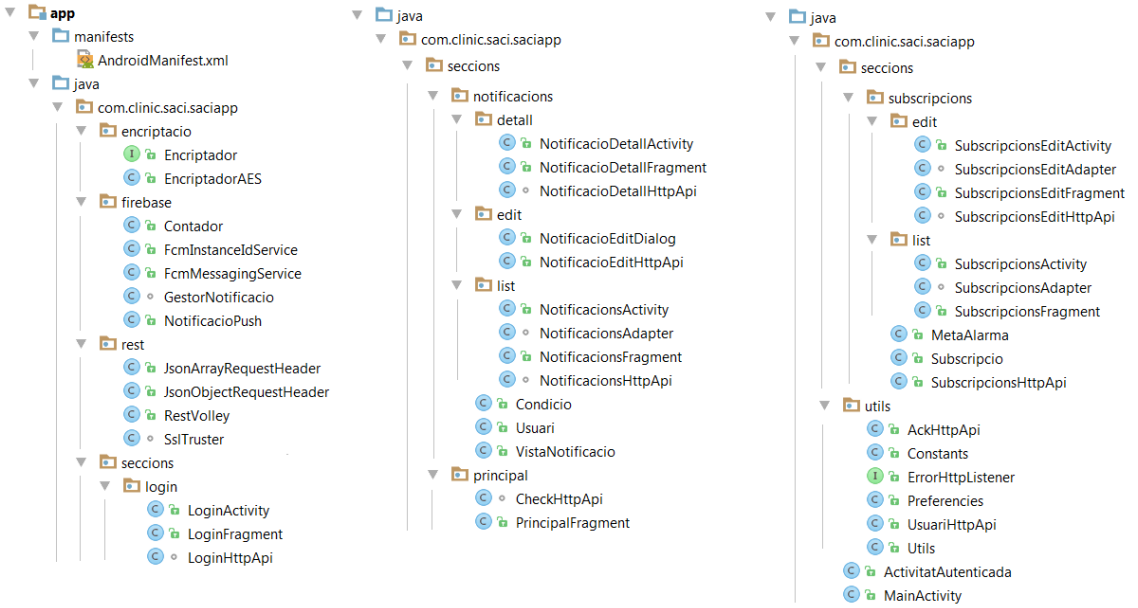

Figure 5.16: Android classes distribution

The total number of classes is almost 50 and the amount of lines is around 5000. This case is similar to the AngularJs code as each class has a different amount of lines depending on its functionality. Instead of modules and controllers, Android uses activities, fragments, adapters, POJOs and other Java elements. The code distribution is similar to the AngularJs application as there are common utility classes used by the main sections classes.

The main difference is that, as this application will be used for receiving push notifications, there are classes for receive, decrypt and show them to the user. These classes are located in the **encriptacio** and **firebase** packages. Once the notification arrives to the device, is **decrypted** using the same algorithm that the server used for encrypting it before sending.

When the notification is decrypted, is sent to a **notification manager** that will show the notification to the user. The notification manager can delay on showing the notification because it waits for more information of the same patient to be received. Thus, if there are two or more notifications from the same patient, can be grouped into a single push notification, being less intrusive than sending them individually.

Common in server and app

```
private static String PASSWORD = [redacted];
private static String ALGORITHM = "AES";
```

```
private Key generateKey() {
    byte[] keyValue = PASSWORD.getBytes();
    Key key = new SecretKeySpec(keyValue, ALGORITHM);
    return key;
}
```

Server side encryption

```
@Override
public String encripta(String text) {
    String encryptedValue = "";
    Key key = generateKey();

    Cipher chiper = Cipher.getInstance(ALGORITHM);
    chiper.init(Cipher.ENCRYPT_MODE, key);
    byte[] encVal = chiper.doFinal(text.getBytes());
    encryptedValue = Base64.getEncoder().encodeToString(encVal);

    return encryptedValue;
}
```

Android side decryption

```
@Override
public String desencripta(String text) {
    String decryptedValue = "";
    Key key = generateKey();

    Cipher chiper = Cipher.getInstance(ALGORITHM);
    chiper.init(Cipher.DECRYPT_MODE, key);
    byte[] decodedValue = Base64.decode(text, Base64.NO_WRAP);
    byte[] decValue = chiper.doFinal(decodedValue);
    decryptedValue = new String(decValue);

    return decryptedValue;
}
```

Figure 5.17: Server and app encryption mechanism

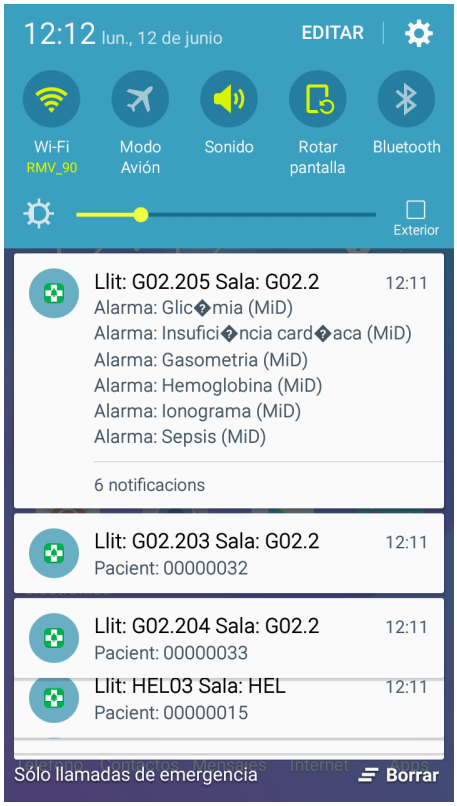

Figure 5.18: Grouped notifications example

6 Validation and testing

This section presents the different tests that have been done for validating that each part of the SCAS was working properly. Each test has been described using a table explaining which test has been done, the expected result of the test and the obtained result from it. While developing, the failure of these tests has helped to correct and solve implementation errors.

6.1 Server security

These tests have been useful to check if the server was offering the expected security level regarding user authentication and HTTPS usage. For checking this, a testing DB with fake data was used. The data set was composed of 7 users (1 admin user and 6 non-admin users), each one having different subscriptions and notifications generated automatically.

Test	Expected result	Test result
Authentication		
Authenticate a user from the HIS into the system by giving the user name and password via HTTP POST method	200 (Ok) HTTP response and JWT token	OK
Authenticate a user who is not from the HIS into the system by giving the user name and password via HTTP POST method	401 (Unauthorized) HTTP response	OK
Request the list of non-admin users via HTTP GET method passing the obtained JWT token in the request header	JSON list containing the information of six users	OK
Request the list of non-admin users via HTTP GET method without passing the obtained JWT token in the request header	401 (Unauthorized) HTTP response	OK
HTTPS usage		
Request any service using HTTP via browser	Automatically the URL switches to HTTPS	OK

Table 6.1: Server security test

6.2 Requesting information to the server

The main goal of these tests has been checking if the methods published via Web Services were working as expected. The data set used for this testing has been the same as the previous section.

Test	Expected result	Test result
Users		
Request the list of non-admin users via HTTP GET method	JSON list containing the information of six users	OK
Request the information of an existing user by giving the user name via HTTP GET method	JSON object containing the information of the requested user	OK
Add a new user by giving its information via HTTP POST method	201 (Created) HTTP response	OK
Update an existing user information by giving its identifier and the information to update via HTTP PUT method	200 (Ok) HTTP response	OK
Delete an existing user by giving its identifier via HTTP DELETE method	200 (Ok) HTTP response	OK
Subscriptions		
Request the list of subscriptions for an existing user by giving the user identifier via HTTP GET method	JSON list of subscriptions where the user is the same for each one	OK
Update the state of some subscriptions for an existing user by giving the user identifier and the list of modified subscriptions via HTTP PUT method	200 (Ok) HTTP response	OK
Notifications		
Request the list of notifications for an existing user by giving the user identifier via HTTP GET method	JSON list of notifications where the user is the same for each one	OK
Request the information of an existing notification by giving the user and notification identifiers via HTTP GET method	JSON object containing the information of the requested notification	OK

Update an existing notification by giving the user and notification identifiers and the modified notification via HTTP PUT method	200 (Ok) HTTP response	OK

Table 6.2: Method publication test

6.3 Client testing

The goal of these tests has been checking if the client applications (web and Android) were showing the expected information. In this case, the tests have been done using the same data set and authenticating with admin and non-admin users. The user section has been tested on web application only, but subscriptions and notifications sections have been tested on both.

Test	Expected result	Test result
Authentication		
Authenticate using existing user name and password	Access to the main menu	OK
Authenticate using non-existing user name and password	Authentication error message	OK
Authenticate without using user name and/or password	Authentication error message	OK
Main menu		
Select subscriptions section	Access to the subscriptions section	OK
Select notifications section	Access to the notifications section	OK
Select users section being admin user (web application only)	Access to the users section	OK
Select users section being non-admin user (web application only)	Stay in the main menu section	OK
Users section (web application only)		
Access the users section	Show the list of non-admin users	OK
Select a user from the list	Show the information of the selected user	OK

Access to the edit user section, modify data and save it	Show the updated information of the user	OK
Access to the edit user section, modify data and discard the changes	Show the original information of the user	OK
Press the delete button and decline confirmation on removing it	Show the information of the user	OK
Press the delete button and accept confirmation on removing it	Show the list of users without the removed user	OK
Subscriptions section		
Access the subscriptions section	Show the list of active subscriptions	OK
Select a meta-alarm filter	Show the active subscriptions for the selected meta-alarm	OK
Access the edit subscriptions section, modify the subscriptions and save the changes	Show the updated list of active subscriptions	OK
Access the edit subscriptions section, modify the subscriptions and discard the changes	Show the original list of active subscriptions	OK
Notifications section		
Access the notifications section	Show the list with all the notifications sent to the user	OK
Select a notification status filter	Show the list of notifications for the selected filter	OK
Select a notification from the list	Show the details of the selected notification	OK
Press the edit button on the selected notification, choose a status and save the change	Show the updated details of the notification	OK
Press the edit button on the selected notification, choose a status and discard the change	Show the original details of the notification	OK

Table 6.3: Client testing

6.4 Data processing

These tests have checked if the server was generating the alarms correctly when processing some patient data. For this, patient data has been generated randomly. The table only shows some of the cases that have been verified as around 200 alarms were generated on each test.

Test	Expected result	Test result
Patient data matches with the required conditions of sepsis risk	A sepsis alarm is generated	OK
Patient data matches with the required conditions of haemoglobin risk	A haemoglobin alarm is generated	OK
Patient data matches with the required conditions of sepsis and haemoglobin risk	Sepsis and haemoglobin alarms are generated	OK
Patient data doesn't match with any alarm defined	No alarms are generated	OK

Table 6.4: Data processing tests

6.5 Push notifications

These tests have been used for checking if the subscriptions were working properly and the notifications were arriving correctly to the devices. For this, using the previous test data (in the cases where an alarm is detected) a notification has been generated and sent through FCM. In order to receive the notifications, two devices have been used. The first device had all the subscriptions activated and the second one was subscribed to the Internal Medicine service alarms.

Test	Expected result	Test result
Generate a sepsis alarm (Internal Medicine service)	Both devices receive the alarm	OK
Generate a haemoglobin alarm (Emergency service)	Only the first device receives the alarm	OK
Generate sepsis and haemoglobin alarms for the same patient	First device shows a grouped notification and second device shows a single notification	OK

Table 6.5: Push notification testing

7 Conclusions

Working in a real environment and trying to include something new to it have been a challenging experience. Stick to times, deal with last-minute changes, depend on users' availability to gather the requirements and wait for the infrastructure to be ready have been the toughest parts of the project. But it has also been a very enriching experience as the learning has been constant and the interest of the people on the project has been very motivating.

Regarding the objectives proposed to achieve, it can be said that all of them have been achieved. The **general objectives** were (1) design and develop the service and (2) implement it in a real-world environment. The service has been designed and developed as a prototype to be tested by the users of the hospital. Currently the Android application has been installed in two devices, the server software has been deployed on a Linux server provided by the hospital in a local network for testing purposes, and the web application is located in the same server and it can be accessed from the hospital internal network. It's also important to say that the tests that have been done until now have been successful.

The **specific objectives**, which were mostly related to learning, seem that have been achieved too. Talking personally, what I have learned is, among other things:

- Improve the knowledge on security techniques, like HTTPS and encryption, and discover new security techniques like Json Web Token. It has made me realise how important security is when dealing with sensitive information.
- Improve Java, Android and AngularJs application development applying new working methodologies. I have been developing for some years, but I always had the feeling that my code could be better. With SOLID principles and code re-factoring I have learned to make a cleaner code and the feeling about my code is better now.
- Use Firebase Cloud Messaging for sending push notifications. It has been very interesting because I didn't know anything about this. Looking for information, trying examples, adapting them to the project and making it work has been very motivating.
- Create better UML diagrams for designing a system correctly. When I was doing the software engineering subject in the university I didn't realise how important is to have a good system design before implementing it. Improving my knowledge on UML has helped me to create the system adjusting it to the needs of the users while having a clear idea on how it should work and which components should have.
- Create PERT and Gantt diagrams to help with project management. I knew about Gantt

diagrams but I didn't know anything about PERT diagrams. Knowing how to design them has helped a lot in the distribution of tasks and times dedicated to each section of the project. Even so, correctly following the temporal distribution of the Gantt chart has not been easy because there have been some unexpected events that have delayed some tasks and made others move ahead.

The people of the hospital have shown interest in the solution as it can be a very useful tool for making some tasks easier to the professionals. Even so, it must be said that this service does not replace any task done by the professionals and, if the service fails, patient safety should not be compromised. It has to be conceived, as mentioned before, as a support tool to help the professionals on doing their tasks.

7.1 Future work

The project has been included in the strategic plan of the hospital. Thus, some actions for short, mid and long term could be proposed in order to make the service a solution to be fully implemented into the hospital. The **short term** actions are related to the integration between SCAS and HIS as the access to real data is not possible right now because there are some services that are being implemented currently. Once these services are ready, it should be possible to continue testing SCAS using real data.

Mid term actions are related to the testing of the SCAS by the users of the chosen medical services. For this, the application should be provided to them in order to get feedback about different aspects. With this feedback, the service should be improved constantly until the users feel comfortable with it. Then, the service should be moved to the production environment and the users should keep using it for checking that it's still working properly. If everything works as expected, then it should be time to expand the SCAS to more medical services and keep working on improvements until it is finally fully integrated into the hospital. Finally, **long term** actions are related to moving the service to a more sophisticated system using, for example, artificial intelligence or adding new functionalities like a meta-alarm generator for the users.

8 References

[1] Martin Alther and Chandan K. Reddy. "Clinical Decision Support Systems". In: *Healthcare Data Analytics*. Chapman and Hall/CRC, 2015. Chap. 19, pp. 625–656. URL: http://www.crcnetbase.com/doi/abs/10.1201/b18588-23.

[2] Wikipedia. *Artifical neural network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.

[3] VisualDx. *VisualDx CDSS*. URL: https://www.visualdx.com/.

[4] Massachusets General Hospital Laboratory of Computer Science. *DXplain CDSS*. URL: http://www.mghlcs.org/projects/dxplain/.

[5] Isabel Healthcare Ltd. *Isabel CDSS*. URL: http://www.isabelhealthcare.com/home/default.

[6] Emma K. Genco. "Clinically Inconsequential Alerts: The Characteristics of Opioid Drug Alerts and Their Utility in Preventing Adverse Drug Events in the Emergency Department". In: *Annals of Emergency Medicine* 67.2 (2016), pp. 240–248. URL: http://www.annemergmed.com/article/S0196-0644(15)01308-6/abstract.

[7] Sara Griesbach et al. "Best Practices: An Electronic Drug Alert Program to Improve Safety in an Accountable Care Environment". In: *Journal of Managed Care & Specialty Pharmacy* 21.4 (2015), pp. 330–336. URL: http://www.jmcp.org/doi/abs/10.18553/jmcp.2015.21.4.330.

[8] Penn Research in Embedded Computing and Integrated Systems Engineering. *Smart Alarms for Medical Device Systems*. URL: http://precise.seas.upenn.edu/research/medical-cyber-physical-systems/decision-support-in-clinical-environments/smart-alarms-for-medical-device-systems/.

[9] Samuel Oloruntoba. *S.O.L.I.D: The First 5 Principles of Object Oriented Design*. URL: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[10] Black Wasp. *The SOLID Principles*. URL: http://www.blackwasp.co.uk/SOLID.aspx.

[11] Clean Coder. *Robert C. Martin (Uncle Bob)*. URL: https://sites.google.com/site/unclebobconsultingllc/.

[12] Google. *Android*. URL: https://www.android.com/intl/es_es/.

[13] Apache. *Apache Tomcat*. URL: http://tomcat.apache.org/.

[14] The PostgreSQL Global Development Group. URL: https://www.postgresql.org/.

[15] Google. URL: https://firebase.google.com/products/cloud-messaging/?hl=es-419.

[16] Change Vision. *Astah - Software Design Tools for Agile teams with UML, ER Diagram, Flowchart, Mindmap and More*. URL: http://astah.net/.

[17] Eclipse. *Eclipse IDE*. URL: https://www.eclipse.org/.

[18] Android. *Android Studio*. URL: https://developer.android.com/studio/index.html?hl=es-419.

[19] Atlassian. *Bitbucket*. URL: https://bitbucket.org/.

[20] Jersey Java. URL: https://github.com/jersey/jersey.

[21] Auth0. URL: https://jwt.io/.

A Annexes

A.1 Acronyms

A

AES Advanced Encryption Standard. 76

C

CAS Clinical Alarms Systems. 10, 13

CDSS Clinical Decision Support Systems. 10, 12, 13

CE Composition Environment. 72

CSS Cascading Style Sheet. 25, 26

D

DB Database. 24, 25, 27, 30–34, 36–40, 44, 49, 51, 72, 76–78, 81, 86

DIP Dependency inversion principle. 18

DSI Departament de Sistemes d'Informació. 4, 5, 11

E

ER Emergency. 19, 22–24, 90

F

FCM Firebase Cloud Messaging. 27, 49, 64, 76, 77, 90, 91

FDP Final Degree Project. 5, 10, 11, 13, 17, 28

G

GUI Graphical User Interface. 72

GW Gateway. 72

H

HCPB Hospital Clínic i Provincial de Barcelona. 4, 5, 11, 19

HIS Hospital Information System. 19, 30, 35, 45–47, 51, 71–73, 79, 86, 92

HTML HyperText Markup Language. 25, 26

HTTPS Hypertext Transfer Protocol Secure. 25, 80, 86, 91

I

ICU Intensive Care Unit. 13

IDE Integrated Development Environment. 28

IM Internal Medicine. 19–22, 90

ISP Interface segregation principle. 18

J

JS JavaScript. 25, 26

JWT Json Web Token. 77, 86, 91

L

LSP Liskov substitution principle. 18

O

OCP Open-closed principle. 18

P

PI Process Integrator. 72, 73

POJO Plain Old Java Object. 77, 79, 84

R

REST Representational State Transfer. 25, 33, 73, 75

S

SCAS Smart Clinical Alarms Service. 30, 35, 36, 45, 64, 65, 71, 73, 74, 76, 77, 80, 81, 86, 92

SRP Single responsibility principle. 18

T

TFG Treball de Final de Grau. 4

TLS Transport Layer Security. 80

U

UML Unified Modelling Language. 28, 30, 32, 35, 36, 39, 40, 45, 51, 52, 55, 59, 63, 66, 91

W

WP Work Packages. 14–17

WS Web Services. 25, 33, 73, 75, 87